(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 491 517 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **91311451.8**

(51) Int. Cl.[5] : **G06F 15/40**

(22) Date of filing : **10.12.91**

(30) Priority : **17.12.90 US 628543**

(43) Date of publication of application :
24.06.92 Bulletin 92/26

(84) Designated Contracting States :
DE FR GB

(71) Applicant : **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor : **Banning, Kenneth Ray**
1701 B West Brake Lane
Austin, Texas 78758 (US)
Inventor : **James, Wendy Sue**
13113 Amarillo
Austin, Texas 78729 (US)
Inventor : **Shih-Gong, Li**
9402 Mystic Oaks Trail
Austin, Texas 78750 (US)
Inventor : **Versteeg, Anton**
309 Ridgecrest Road
Georgetown, Texas 78628 (US)

(74) Representative : **Tomlinson, Kerry John**
Frank B. Dehn & Co. European Patent Attorneys Imperial House 15-19 Kingsway
London WC2B 6UZ (GB)

(54) Tree structure representation of an SQL clause.

(57) A system and method for graphically representing a WHERE or HAVING clause of an SQL query directed to a relational database. Logical operators are defined and linked to predicates using a tree structure format (23). The tree is not binary in character and consequently exhibits a 1 :1 relation between the predicates and tree leaves. The tree structure representation provides intuitive feedback to the user defining the query. Preferably the SQL format (24) and tree structure graphic format (23) queries appear simultaneously on the video display of the computer system used to define the query.

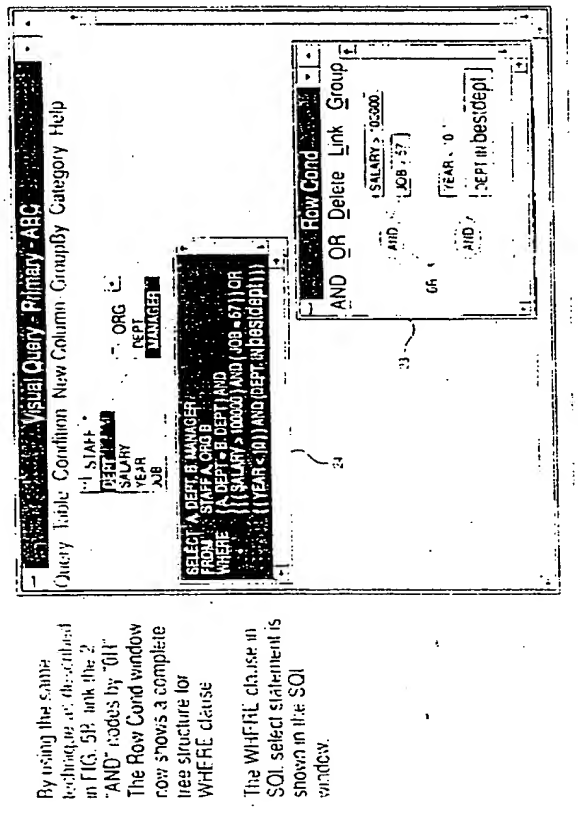


FIG. 5E

EP 0 491 517 A2

The present invention relates in general to an interface between a human user and a computerized relational database. More particularly, the invention is directed to a system and method for creating relational database queries using graphical tree structure depictions, to express WHERE and HAVING clauses.

The entry and retrieval of data via interactive terminals is now a common practice. Large volumes of data stored on magnetic disks, optical disks, or other contemporary non-volatile storage media, are accessed locally or through networks using terminals or functionally equivalent workstations or personal computers. Such prevalent uses of databases has created an environment in which the level of data processing skill possessed by an average user is disproportionally low in relation to the complexity of the hardware and software functions available to interact with the database. This is particularly true for relational databases, where the information content is significant yet the query mechanisms for defining the desired information involve esoteric data definitions and groupings. Though the technology of accessing information has progressed beyond the use of structured query language (SQL) statements to the use of query by example (QBE) and prompted query techniques, there remains a need for further simplification and ease of use.

Relational database query expressions with WHERE and HAVING clauses are well known to those who have a working knowledge of SQL. However, the expressions can become quite lengthy and lose their intuitive meaning as the number of predicates and relationships in the query increases. Thus, there has evolved a need for a system and method which allows a moderately skilled user to create and manipulate WHERE and HAVING clauses in relational database queries.

U.S. Patent No. 4,506,326 teaches the use of query by example (QBE) to formulate structured query language (SQL) syntax. Unfortunately, the QBE approach is limited in application and does not provide in graphical or other pictorial form an intuitive representation of what the expression represents in relation to the information in the database.

The use of tree-like graphics to represent Boolean logic is not new. Tree representations are used in hypercard type applications and artificial intelligence depictions. The use of Boolean factor trees to represent SQL statements is discussed in an article entitled "Method of Detecting Atomic Boolean Factors" as appeared on pages 368-374 of the IBM Technical Disclosure Bulletin, Vol. 32, No. 5B, dated October 1989. Therein it is noted that WHERE and HAVING clauses of SQL statements can be represented by Boolean factor trees. However, it is also noted that Boolean factor trees are binary. Boolean factor trees have a structure which loses intuitive worth when the number of predicates associated with a logical operation exceeds two.

A representative relational database is a part of the OS/2 (trademark of IBM Corporation) Extended Edition operating system program, available from IBM Corporation, as particularly suited for use in a PS/2 (trademark of IBM Corporation) workstation. A preferred configuration for the workstation includes a graphics display, a keyboard, a mouse and cabinet resident hard disk drive. A typical workstation would also include communication and networking cards suitable to access remote databases or databases resident on host computers. In the context of such configurations, there exists a need for a system and method by which a user of moderate database expertise can define, manipulate and utilize a relational database query including those constructed with relatively complex WHERE and HAVING clauses.

The invention presented herein defines systems and processes for using graphical representations of WHERE and HAVING clauses to define and intuitively represent queries for relational databases. The invention contemplates the use of tree structures having logical operators, such as OR and AND, to link predicates. The use of graphics and fundamental logical operators provides the user with a pictorial representation of the expression being created or modified. The tree structure has a form which is amenable to automatic conversion into SQL using commonly known parsing techniques.

The intuitive quality of the graphical representation according to the present invention is attributable to the use of an SQL related tree structure which has resources to handle two or more predicates for each logical node. This tree representation provides a 1:1 mapping between predicates and the leaf nodes of the tree. In contrast, conventional Boolean factor trees constrain the number of permissible leaves to two per logical node, creating structures which commonly fail to have a 1:1 mapping. The mapping as provided by the tree structure of the present invention retains the intuitive link between the graphical representation and the query objective sought by a user of moderate skill when interfacing to a relational database.

These and other aspects of the invention will be understood and appreciated with greater specificity upon considering the detailed description of a preferred embodiment which follows by way of example only and with reference to the accompanying drawings.

Brief Description of the Drawings

Fig. 1 is a schematic block diagram of a workstation using a graphics media to define a SQL query.

Fig. 2 illustrates a prior art Boolean factor tree.

Fig. 3 illustrates a tree according to the present invention.

Figs. 4A-4C schematically illustrate windows used for defining a query.

Figs. 5A-5I schematically illustrate windows for refining and expanding the query.

Figs. 6A-6J schematically illustrate flow diagrams of the program operation.

Fig. 1 illustrates the operating environment to which the invention pertains. As such, the invention preferably includes and utilizes a PS/2 workstation having a cabinet 1 (with resident processor, memory, and disk storage), keyboard 2, user interactive mouse-type control 3, and graphics quality video display 4. The workstation preferably also includes within cabinet 1 a hard or optical disk non-volatile memory, a local area network board, and a communications board. The non-volatile storage provides resident database capability, while the boards provide communication interfaces to network or host resident databases. The preferred operating system for the workstation is the OS/2 Extended Edition. Such operating system provides the user with resources to create and interact with the OS/2 Extended Edition relational database using SQL statements.

The WHERE and HAVING query clauses, which are frequently used in SQL statements, have proven to be difficult for moderately skilled users to create and understand when formulated in SQL language. To a significant extent the difficulty is attributed to the structure of such SQL clauses, in that the clauses involve strings of text grouped by multiple parentheses.

Boolean factor trees have been used in compilers to parse program expressions into forms amenable for manipulation by digital computers. Fig. 2 illustrates a Boolean factor tree for the expression (A AND B) OR (C AND D). In keeping with the convention of the Boolean factor tree, each logical operator relates only two predicates. Thus, the leaves of the tree, such as elements A, B, C and D, are connected in binary pairs to each logical operator, here the operator OR. This binary character is reflected at each level, in that there are two OR nodes 6 and 7 for AND node 8 as well as two OR nodes 9 and 11 for AND node 12. The same binary character exists between the two AND nodes 8 and 12 and their relating logical operator in node 13. As a consequence of its binary structure, the Boolean factor tree requires leaf node predicates to be repeated in the depiction, such as predicate A both in blocks 14 and 16. Such predicates only appear once in the logical expression. The unfortunate outcome of using such a tree is that the intuitive link between the meaning of expression (A AND B) OR (C AND D) and any meaning acquired from its pictorial representation is effectively nonexistent. The use of Boolean factor trees to represent WHERE and HAVING clauses within an SQL statement are considered within the aforementioned IBM Technical Disclosure Bulletin.

The invention provides a tree structure which maintains the intuitive link between the graphical representation and the desired relationship as specified in the SQL WHERE and HAVING clauses. To further assist the user, the invention also provides a system and method which translates between SQL clauses and the tree structures to allow the user to concurrently perceive through multiple media, i.e., tree graphics and SQL text, the effects of any changes to the WHERE and HAVING clauses. These features are preferably provided in the context of the workstation depicted in Fig. 1, using a window such as 13 on video display screen 4.

Fig. 3 illustrates a tree structure according to the present invention for representing the logical arrangement (A OR B OR C) AND (D AND (E OR F)). In contrast to the Boolean factor tree, note that three leaves, A, B and C, extend from OR node 16. As such, the tree is no longer binary. Note also that each leaf predicate appears only one time within the tree. The benefits of such a tree structure for expressing a WHERE or HAVING clause become even more evident upon comparing the intuitive relationship acquired from considering the pictorial representation to the logical relationship defined by the Boolean expression. The relationships between the predicates A-F are fairly clear even to a user of moderate skill with query formulation. Foremost, there exists a singular translation between the graphical representation and the SQL statement which it represents.

The preferred implementation of the invention is presented by way of simple examples in Figs. 4A-4C, and by extension into refined examples in Figs. 5A-5I. Fig. 4A shows at 14 the primary window of a query as would appear on workstation screen 4 (Fig. 1), and includes therein subwindows individualizing the columns to which the query is to be directed. In this case the columns are "STAFF" and "ORG", respectively at reference numerals 16 and 17. Also present in window 14 is subwindow 18, showing the SQL statement being defined by the query. In subwindow 19 the predicates of the query are defined and related by logical operators "AND" and "OR". The images which appear in Figs. 4A-4C depict the succession of operations and effects which define a relatively simple query. Fig. 4A shows the specification of an argument, defining that the desired information relates to "STAFF" having a "SALARY" greater than 100,000. Fig. 4B illustrates the effects of linking the "SALARY" argument with a predicate specifying "YEARS" less than 10 via the logical AND condition. Fig. 4C depicts the further inclusion of a "JOB" category 57, again relating through the logical AND operator. The tree representation is created by user manipulation of a mouse and keyboard and is visually portrayed in subwindow 21. The corresponding SQL statement appears in block 22.

The pictorial and SQL WHERE clause defined in Fig. 4C can be created, according to the preferred practice of the invention, by manipulation of either the SQL statement in subwindow 22 or the tree structure in subwindow

21. Where the user has the ability to specify the SQL statement directly in subwindow 22, the preferred practice of the invention generates its graphical equivalent by tree representation in subwindow 21. This is accomplished by applying existing parsing techniques to the SQL statement and thereby derive the predicates and relating logical operators.

The features of the invention are most valuable to a moderately skilled user, one more likely to define the query containing WHERE or HAVING clauses in subwindow 21 using the graphical method. Namely, the user creates a tree structure representation of the desired WHERE or HAVING clauses by entering predicates, selecting logical operators, and relating the predicates to the operators. The tree structure so defined is used by the system to generate the equivalent SQL statement in subwindow 22. It is worth noting that the user is not burdened with the particulars of the SQL language, but rather can depend upon the intuitive characteristics represented by the tree depiction of the clause. It should also be apparent that a binary tree of the Boolean form would be incapable of readily representing the three predicates in subwindow 21, which combination is needed to represent the clause.

The succession of Figs. 5A-5I illustrate an extension of the earlier example to more elaborate query and modification practices. As shown in Fig. 5A, the three previously defined predicates are unlinked for further refinement. Fig. 5B shows the linking of two predicates through a logical AND operator, and the further addition of another predicate to reach the stage depicted in Fig. 5C. The second set of two predicates is then linked as appears in Fig. 5D using a logical AND operator. The concluding clause is depicted by tree structure in Fig. 5E, therein showing a logical combination of the two AND operators via a common OR operator. The SQL statement corresponding to the tree representation appears as a classic SQL statement in subwindow 24.

Fig. 5F illustrates the ease with which a graphical interchange of logical operators can be accomplished in the context of the tree structure when the user has doubts about one or more logical links, or desires to see the effects of rearranging the tree. It should be apparent that the query expressed by the tree in subwindow 23 of Fig. 5E is readily understood to differ from the argument as represented by the tree in subwindow 26 of Fig. 5F. In contrast, note the significant mental juggling required to appreciate the differences when comparing the SQL statements as appear in subwindow 24 of Fig. 5E and subwindow 27 of Fig. 5F.

Fig. 5G illustrates that a user can readily add new predicates to an existing WHERE clause tree structure. Fig. 5H illustrates the ease with which the newly defined predicate is linked to a previously specified logical operator, in this case OR, and continues to allow the user to retain intuitive understanding of the complex WHERE clause being defined. This is in distinct contrast to the difficulty a user experiences in trying to understand the SQL based WHERE expression set forth above, given that the user must rely upon deciphering the multiplicity of parentheses to understand the interaction of the logical operators and predicates.

Fig. 5I further illustrates the ease with which the predicates in Fig. 5H can be relocated in the tree structure. Again, note the contrast between attempting a comparison of the tree representations to the equivalent SQL representations of the WHERE clause arguments as appear in the respective subwindows of Figs. 5H and 5I. It is this ability of the tree arrangement to convey an intuitive representation of the relationships between predicates which makes the invention particularly useful to users of moderate skill.

Figs. 6A-6J present, in object oriented format, flow diagrams relating significant operations performed by the present embodiment of the invention. The symbol "A" within a circle represents a state within which the system, program or process is awaiting an input from the user. The selection provided by the user thereafter defines which of the sequences, Figs. 6B-6J, are undertaken in generating or modifying the tree structure or its functional equivalent SQL statement.

Pseudo code, from which source code can be derived, for a program for implementing the present invention is set forth below. The cases represented are consistent with the selections in the flow diagrams in Figs. 6A-6J.

```
/* Initialization                */
    Predicate_Count = 0
    Row_Cond_Window is hidden
    Next_Predicate_No = 1
    Group_Status = off      /* Group Selection off   */
    GroupList is empty
    OP_Change = off         /* Operator Node Change Off  */
    Link_Status = off       /* Linking a node to another */
                            /* node is off               */
    Default = on            /* Default is on    */
    SELECTED = null         /* Nothing being selected  */
/*****************************/
/*    Process Begins         */
```

```
/*****************************/
CASE  New_Predicate_Defined
    No_of_Predicate = No_of_Predicate + 1
    Idx = Next_Predicate_No
    Next_Predicate_No = Next_Predicate_no + 1
/* Put the text format of the new predicate into TEXT */
    TEXT(Idx) = New_Predicate(text)
/* Find the (x,y) coordinates for displaying this  */
/* predicate in Row Condition Window               */
    X(Idx) = POSITION(X)
    Y(Idx) = POSITION(Y)
    if Row_Condition_window is hidden
       display Row_Condition_Window
    endif
    display the new predicate in Row_Condition_Window at
      (X(Idx),Y(Idx))
    if Default = on
       if No_of_Predicate = 2
          Create an OP node "AND"
          Link all existing predicates with this "AND" OP
            node
          Mark this OP node as Default_Node
       else
          Link this new predicate with the Default_Node
       endif
    endif
    Group_Status = off
    GroupList = null
    Link_Status = off
    OP_Change = off
    SELECTED = null

CASE  Single_Click_on_Predicate
    if Group_Status = off
       Reset any highlighted predicate or OP node to
normal
       display
```

```
        endif
        Highlight display the clicked predicate
        if Group_Status = on
           Add this predicate's Idx into GroupList
        endif
        Mark this predicate as SELECTED
        Link_Status = off
        OP_Change = off

Case  Double_Click_on_Predicate
        From the predicate's (X,Y) coordinates find the Idx of
          this predicate
        Highlight display this predicate
        Get TEXT(Idx)
        Parse TEST(Idx) to generate the LeftSide, CompOP,
          RightSide
        Display the dialog box showing this predicate's
LeftSide
          in an text entry field, mark the CompOp in a list of
          all allowed comparison operators, the RightSide in a
          text entry field with label "Value" or a text entry
          field with label "Subquery" if the RightSide is a
          subquery name.  This can be judged from CompOp.
        if Dialog Box is committed
           Collect new LeftSide, CompOp, RightSide to build
             TEXT(Idx)
           Update the display of this predicate
           Update the corresponding SQL statement
           Update the SQL_Window display
        endif
        Reset display to normal
        Group_Status = off
        GroupList = null
        Link Status = off
```

```
    OP_Change = off
    SELECTED = null

CASE  Single_Click_on_OpNode /* OP nodes are "AND"/"OR" */
    if Link_Status = on
       Reset the highlighted node to normal display
       Put the information about the two linked nodes into

         the data structure for the tree representation
       Display a link line between the two nodes
       Link_Status = off
    else
       Highlight display the clicked OP node
       Mark this OP node as SELECTED
       OP_Change = on
       Group_Status = off
       GroupLIst = null
       Link_Status = off
    endif

CASE  DeleteAction /* Delete key pressed or Delete menu */
                   /* selected */
     if SELECTED not equal to null
        Display the deletion confirmation box
          if confirmed
            delete the selected OP node or predicate
            erase its display
            if predicate deleted
               No_of_Predicate = No_of_Predicate - 1
               Remove information of this predicate
            endif
            if OP node
               Remove information of this OP node
            endif
            Update the SQL statement
            Update the SQL_Window display
            if SELECTED = Default_node
```

```
            Update the SQL statement
            Update the SQL_Window display
            if SELECTED = Default_node
               Default = off
            endif
          endif
      endif
      Group_Status = off
      GroupList = null
      Link_Status = off
      OP_Change = off
      SELECTED = null

CASE  ANDmenu     /* AND menu selected    */
      if OP_Change = on
         Change the SELECTED OP node to be "AND"
         Change the Row_Condition_Window display
      endif
      if Group_Status = on
         Link all items in the GroupList together with "AND"

           relationship
         Create a new "AND" OP node
         Display the new "AND" node with lines linked with
           nodes in the GroupList
      endif
      Update the SQL statement
      Update the SQL_Window display
      OP_Change = off
      Group_Status = off
      GroupList = null
      SELECTED = null
      Link_Status = off

 CASE  ORmenu     /* OR menu selected   */
      if OP_Change = on
         Change the SELECTED OP node to be "OR"
```

```
        Change the Row_Condition_Window display
      endif
      if Group_Status = on
         Link all items in the GroupList together with "OR"
           relationship
         Create a new "OR" OP node
         Display the new "OR" node with lines linked with
           nodes in the GroupList
      endif
      Update the SQL statement
      Update the SQL_Window display
      OP_Change = off
      Group_Status = off
      GroupList = null
      SELECTED = null
      Link_Status = off

CASE  GROUPmenu   /* GROUP menu selected    */
      Group_Status = on
      GroupList = null
      OP_Change = off
      SELECTED = null
      Reset any highlighted display to normal display

CASE  LINKmenu    /* LINK menu selected     */
      if Group_Status = off and OP_Change = off and SELECTED

           is not null
         Link_Status = on
      else
         Group_Status = off
         GroupList = null
         OP_Change = off
         SELECTED = null
         Link_Status = off
      endif
```

The invention thus provides a user of a relational database with an intuitive and user friendly tool for operating and manipulating queries containing WHERE or HAVING clauses using graphical representations. The intuitive feedback provided by the tree structure makes it easy to understand the substance of the database query being formulated. The concurrent presence of the tree structure and SQL statement allows the user to

manipulate either in creating the desired expression. Such concurrency also provides a training tool for users of SQL statements, in that the meanings underlying SQL statements are immediately available in graphical form as feedback. The contemplated tree structure is not binary in character, and as such is not restricted to two predicates per logical operator. Similarly, the tree structure leaves are related to the SQL predicates on a 1:1 basis. The combination thus provides the intuitive link needed by a contemporary user of relational databases.

Though the invention has been described and illustrated by way of specific embodiments, the apparatus and methods should be understood to encompass the full scope of the features defined by the claims set forth hereinafter.

Claims

1. A system for accessing information in a relational database, comprising:
   means for a user of a computer to define a query in a form suitable for accessing the relational database; and
   means for graphically depicting, by tree structure supporting three or more predicates per logical operator, an argument of the defined query.

2. The system recited in Claim 1, wherein the query is an SQL statement containing a WHERE or HAVING clause.

3. The system recited in Claim 1 or 2, wherein the number of tree structure leaves matches the number of predicates in the query.

4. The system recited in any preceding Claim, wherein the logical operators are AND and OR functions.

5. The system recited in any preceding Claim, wherein the means for a user to define a query includes both a keyed alphanumeric entry device and a user actuated graphic control device.

6. A method for accessing information in a relational database of a computer system, comprising the steps of:
   graphically defining, as a first part of a query, one or more logical operators;
   graphically defining, as a second part of the query, three or more predicates; and
   graphically defining, as a third part of the query, links relating three or more predicates to a common logical operator.

7. The method recited in Claim 6, wherein the links represent the argument of a WHERE or HAVING clause in the query.

8. The method recited in Claim 6 or 7, wherein the logical operators are AND and OR functions.

9. The method recited in Claim 6, 7 or 8, including the further step of directly relating the graphically depicted query parts to an SQL statement in the computer system.

10. A system for generating data defining a query in a form suitable for accessing a database, comprising an operator input means, a memory means for storing data determined by input from said input means, means for displaying an image corresponding to the data stored in said memory means, said system being arranged to enable an operator to graphically define on said displaying means a logical query in the form of an image of the query having one or more logical operators, three or more predicates and links relating three or more predicates to a common operator, the system further comprising a processor means for processing data in said memory means to generate further data representing said imaged query in a form suitable for accessing a database.

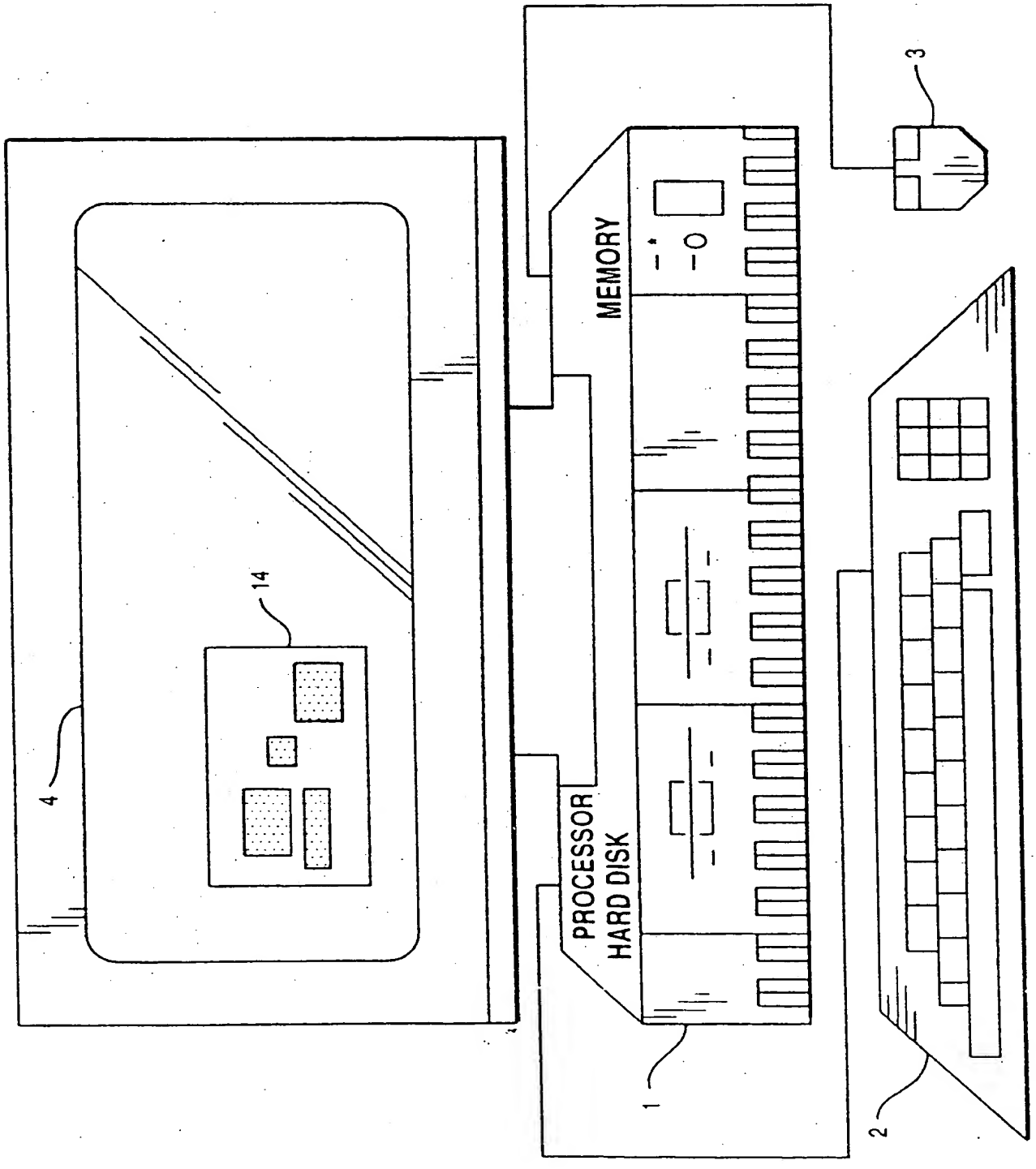


FIG. 1

BOOLEAN FACTOR
TREE FOR
(A AND B) OR (C AND D)

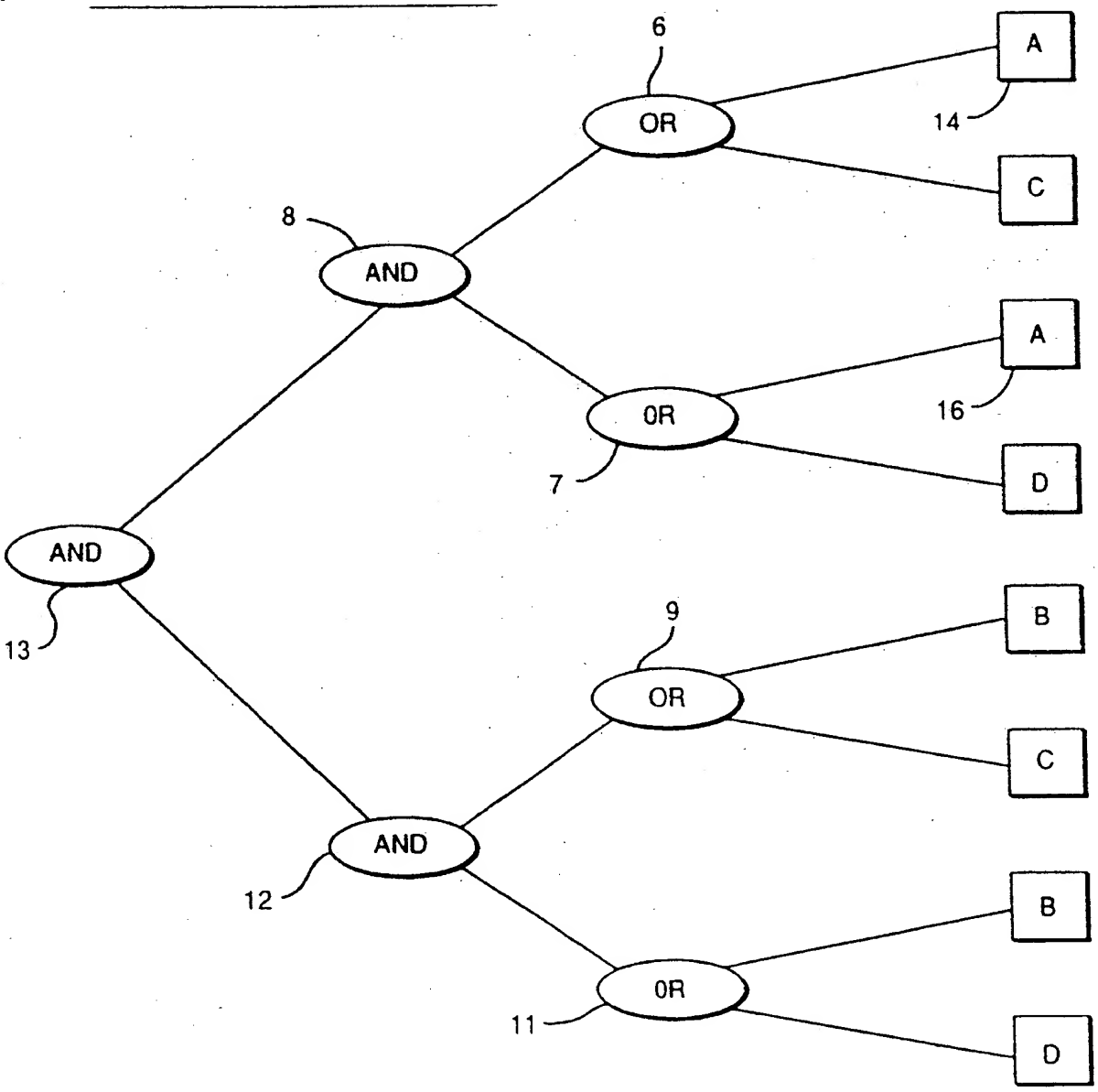


FIG. 2

PRIOR ART

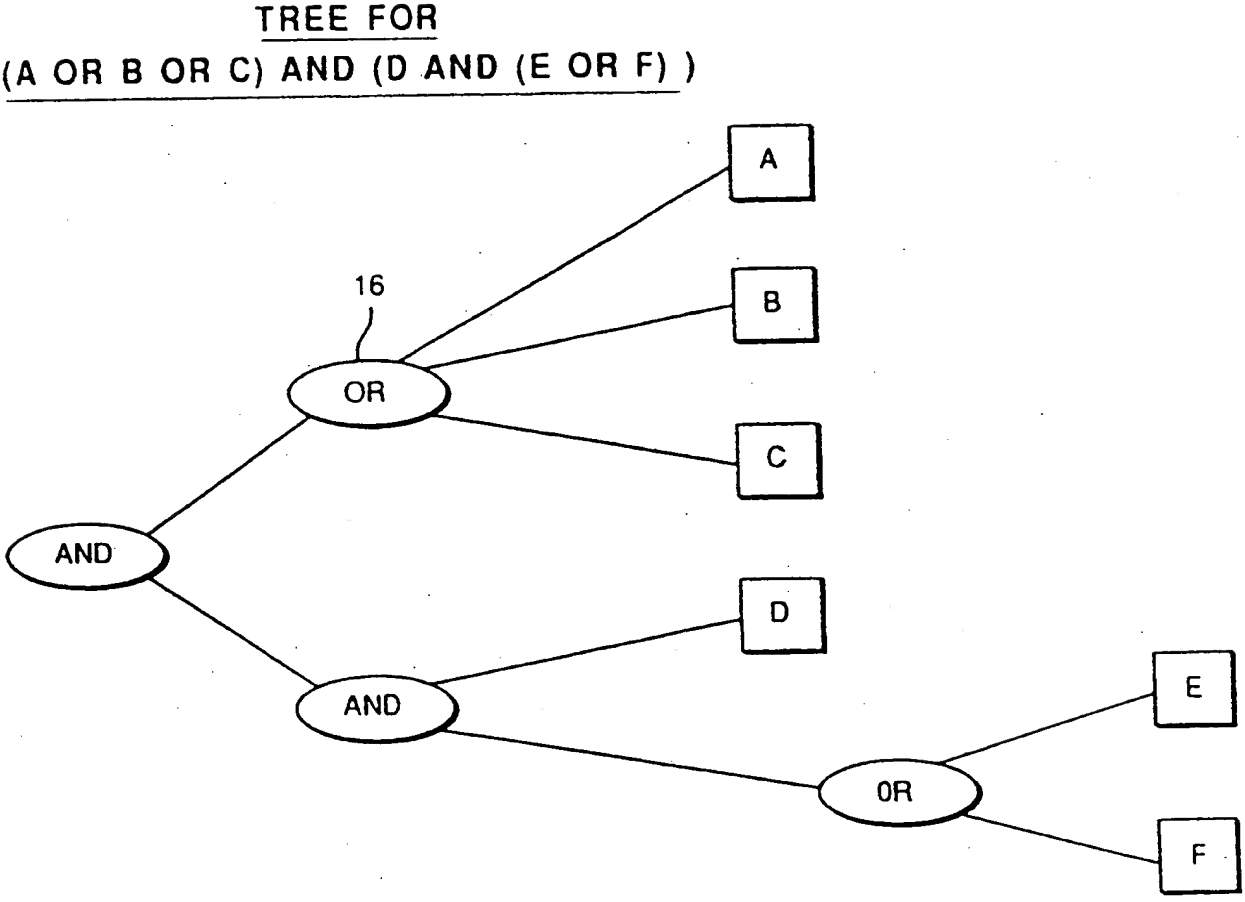


FIG. 3

(1) DEPT of STAFF table and MANAGER of ORG table are selected columns.

(2) STAFF and ORG are joined by their DEPT columns, it is presented by a straight line connecting the 2 columns. The join condition is expressed as (A. DEPT = B. DEPT) in WHERE clause.

(3) A predicate "SALARY > 100000" is defined and it is the only one defined so far.

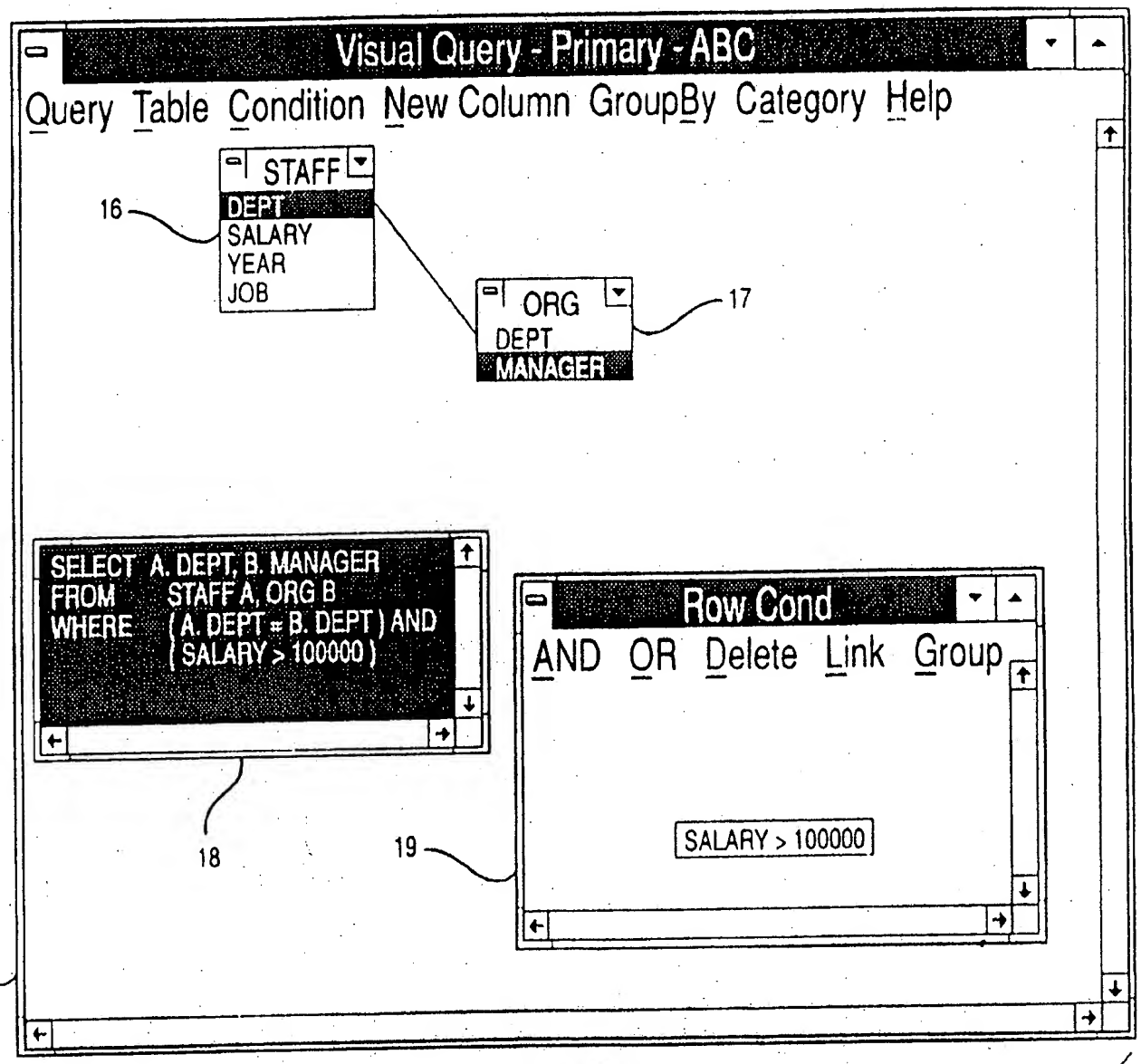


FIG. 4A

EP 0 491 517 A2

(1) A new predicate "YEAR < 10" is defined.

(2) The 2 defined predicates are linked together by "AND" as default.

(3) The WHERE clause now becomes:

(A. DEPT = B. DEPT)

The Join Condition

AND

((SALARY > 100000) AND (YEAR < 10))

The Row Condition

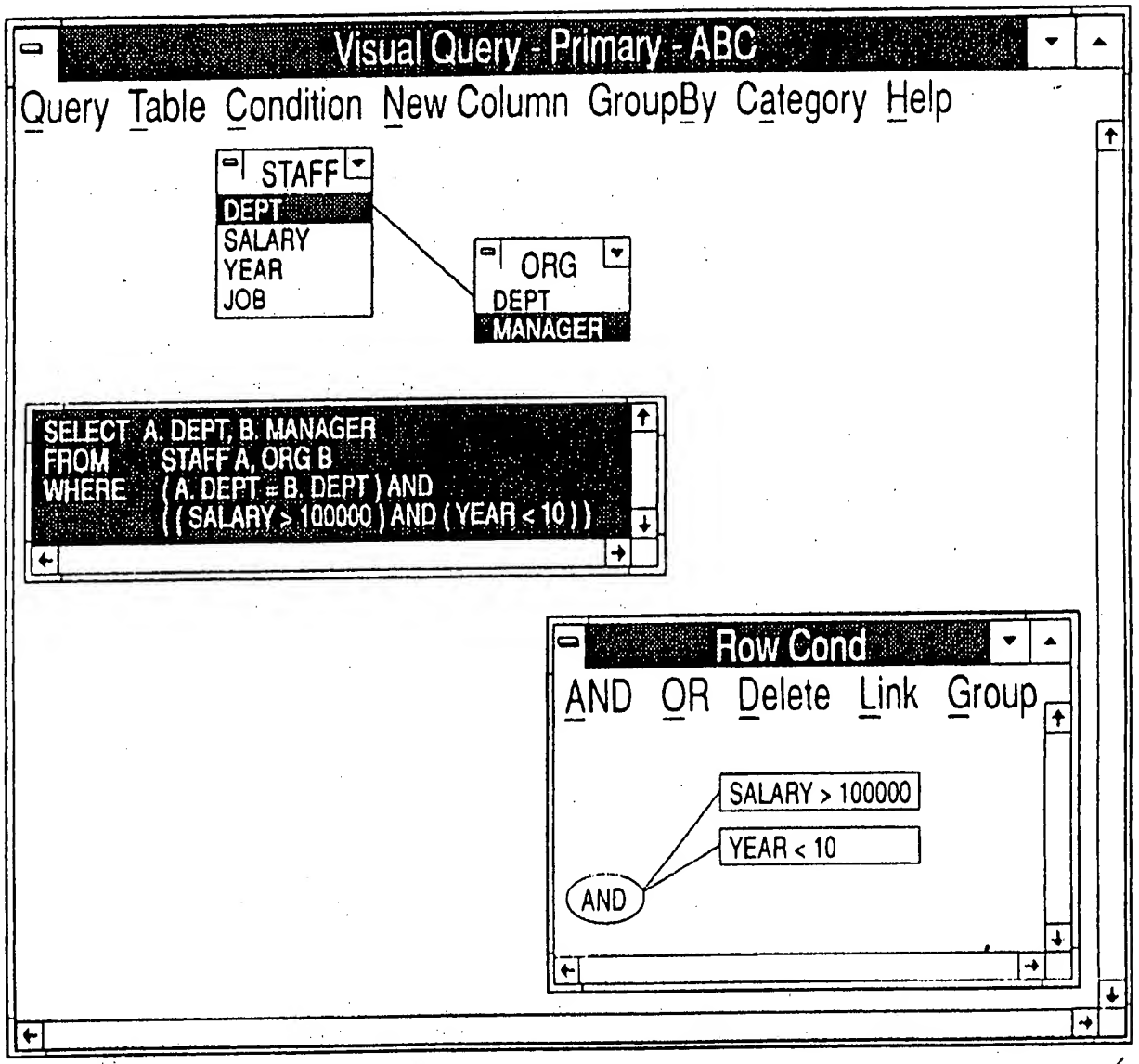


FIG. 4B

EP 0 491 517 A2

(1) A new predicate "JOB = 57" is defined and linked with the previously defined predicates by "AND".

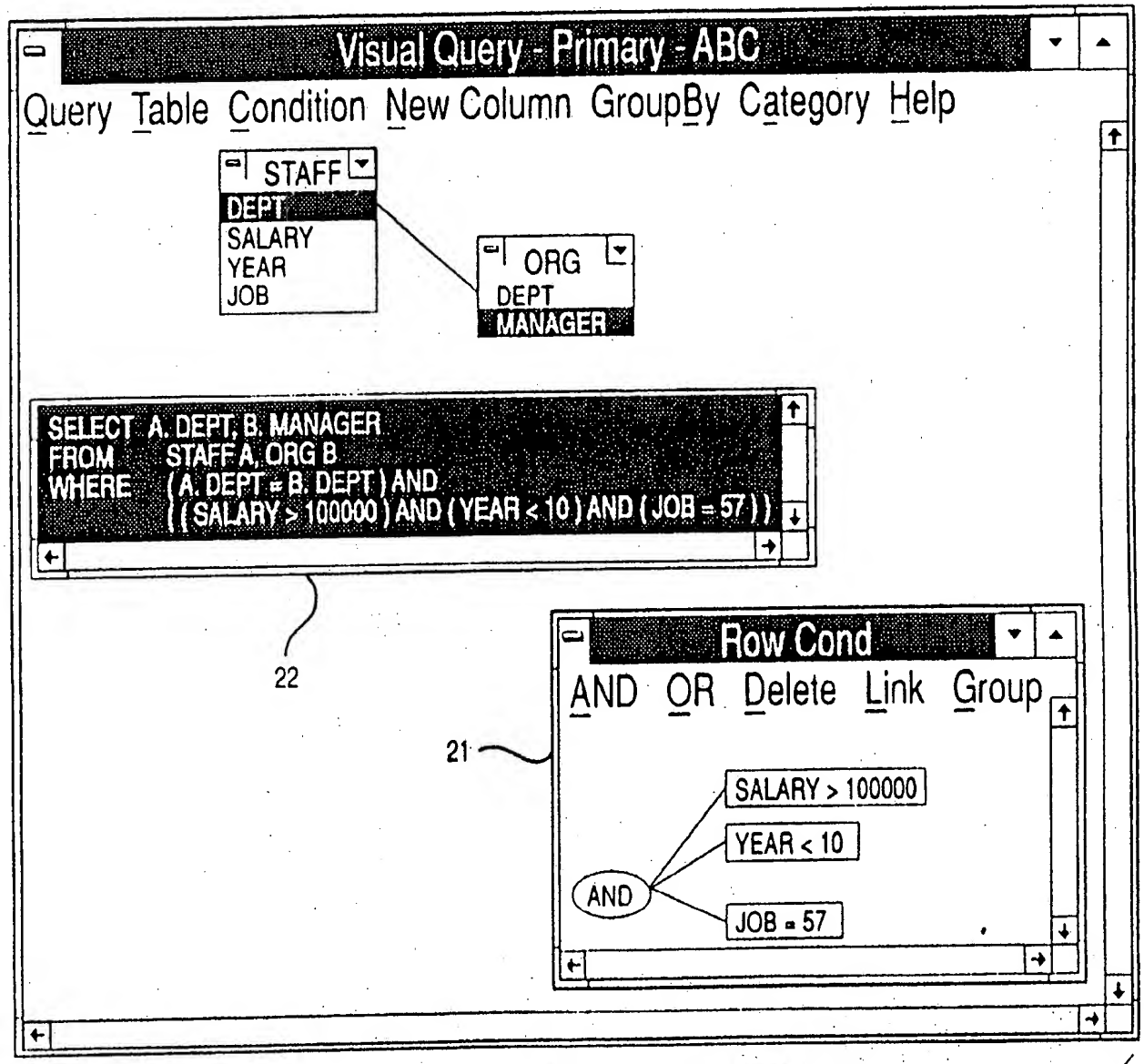


FIG. 4C

(1) To construct a new logical relationship to link the defined predicates, the first step is to delete the "AND" and leave all the predicates un-linked.

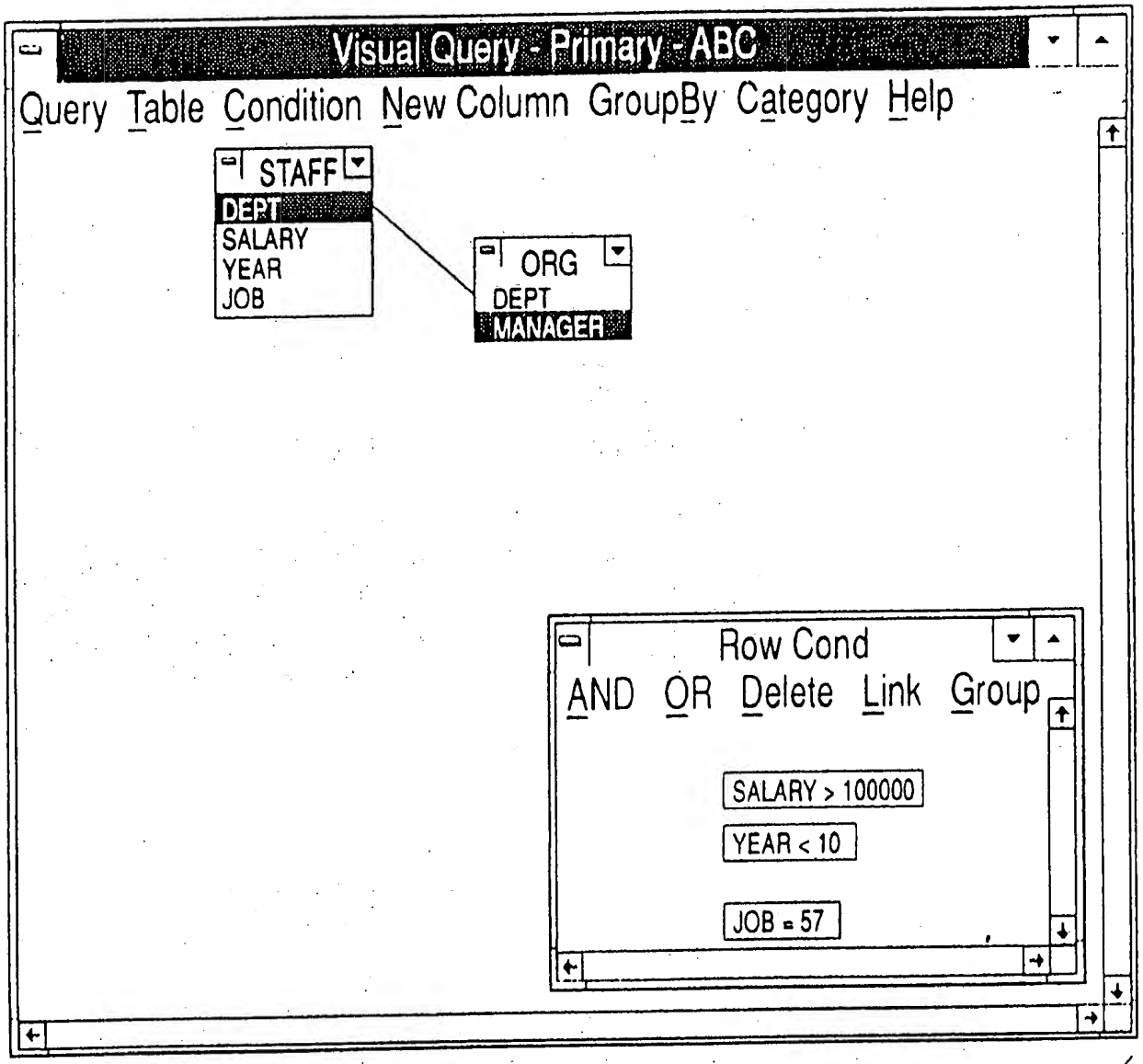


FIG. 5A

EP 0 491 517 A2

(1) To link "SALARY > 100000" and "JOB = 57" by "AND", the operations are:

(a) Click on Group, select "SALARY > 100000" by clicking on it, select "JOB = 57" by clicking on it

(b) Click on "AND'

(2) Then, the screen display will be changed as shown above. Notice that predicates have been repositioned.

—— Another way to operate (1) is (1)':

(1)' Extended selection on "SALARY > 100000", "JOB = 57" by SHIFT + mouse clicking.

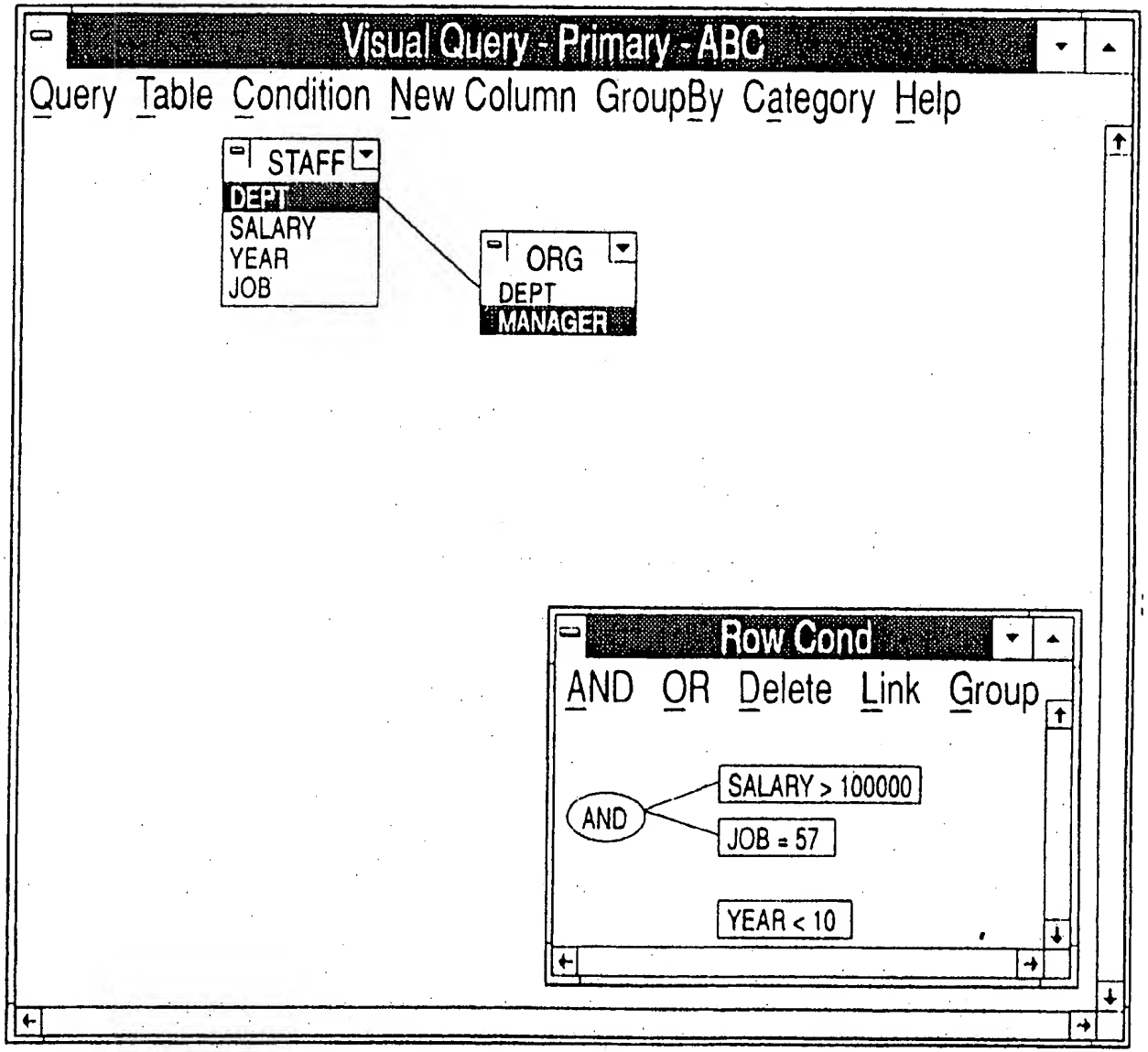


FIG. 5B

(1) A new predicate "DEPT IN bestdept" is defined and the screen display is changed to show this new predicate in the Row Cond window.

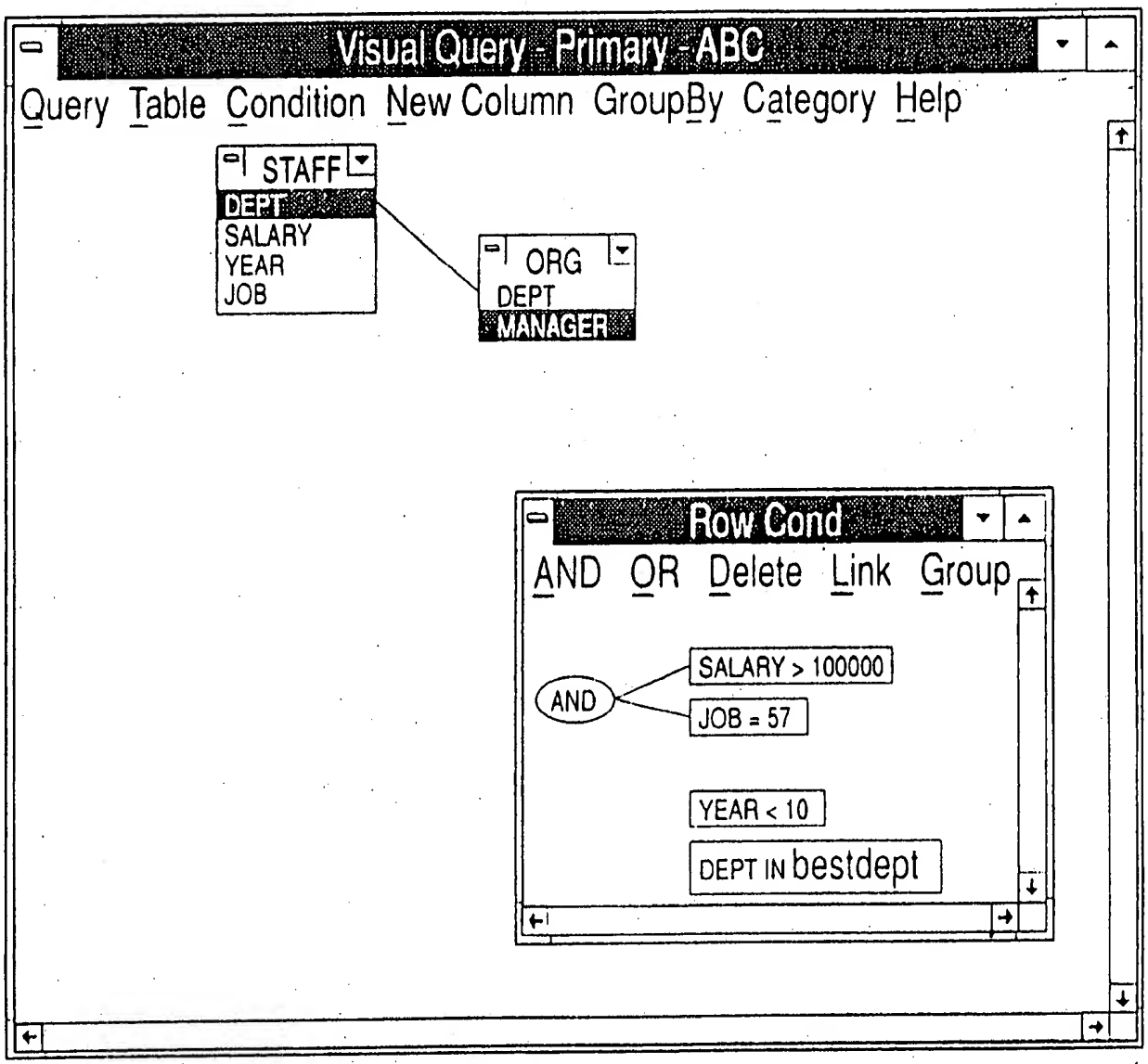


FIG. 5C

EP 0 491 517 A2

(1) By using the same technique as described in FIG. 5B, "YEAR < 10" and "DEPT IN bestdept" are linked by "AND".

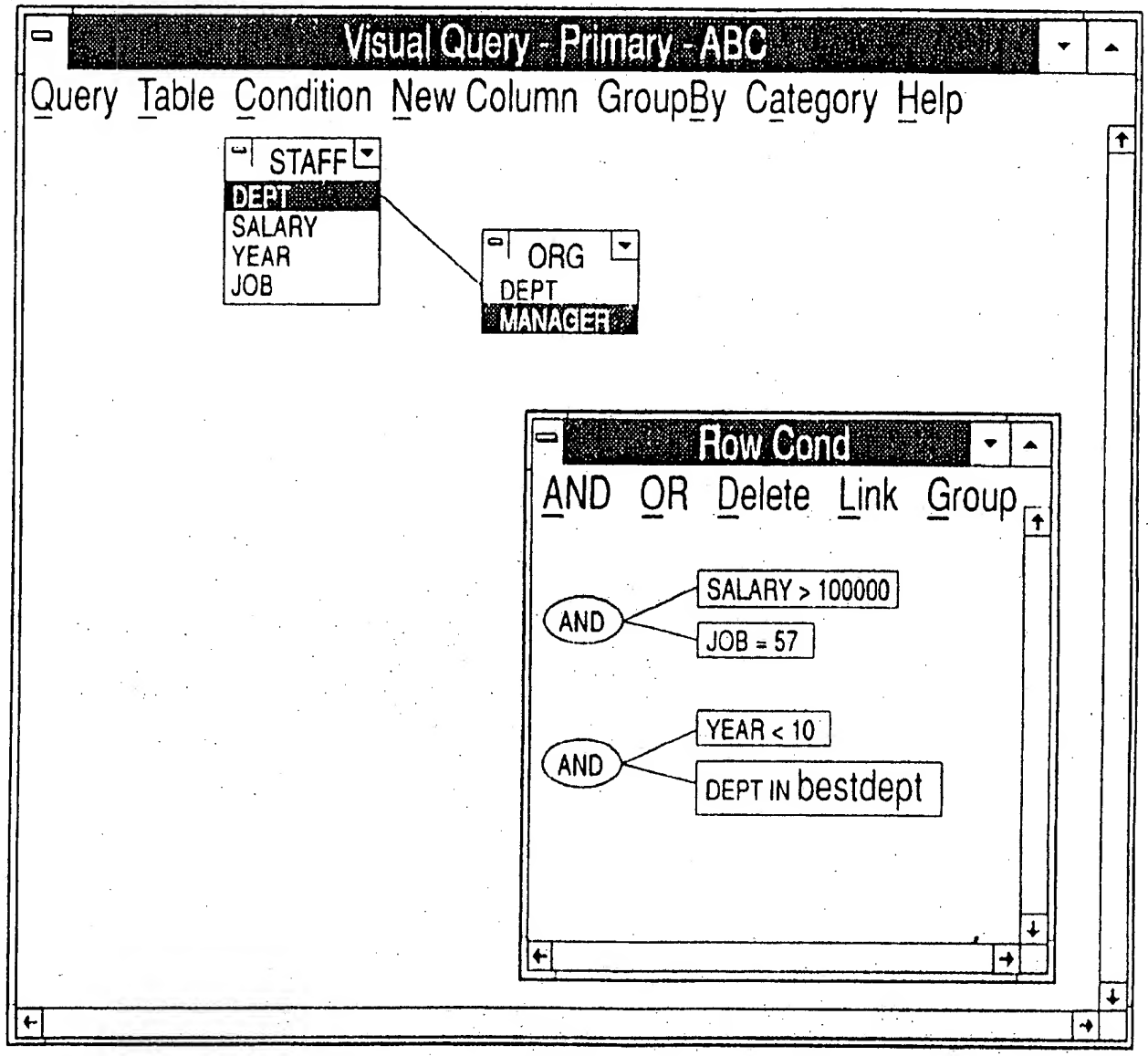


FIG. 5D

EP 0 491 517 A2

(1) By using the same technique as described in FIG. 5B, link the 2 "AND" nodes by "OR". The Row Cond window now shows a complete tree structure for WHERE clause.

(2) The WHERE clause in SQL select statement is shown in the SQL window.

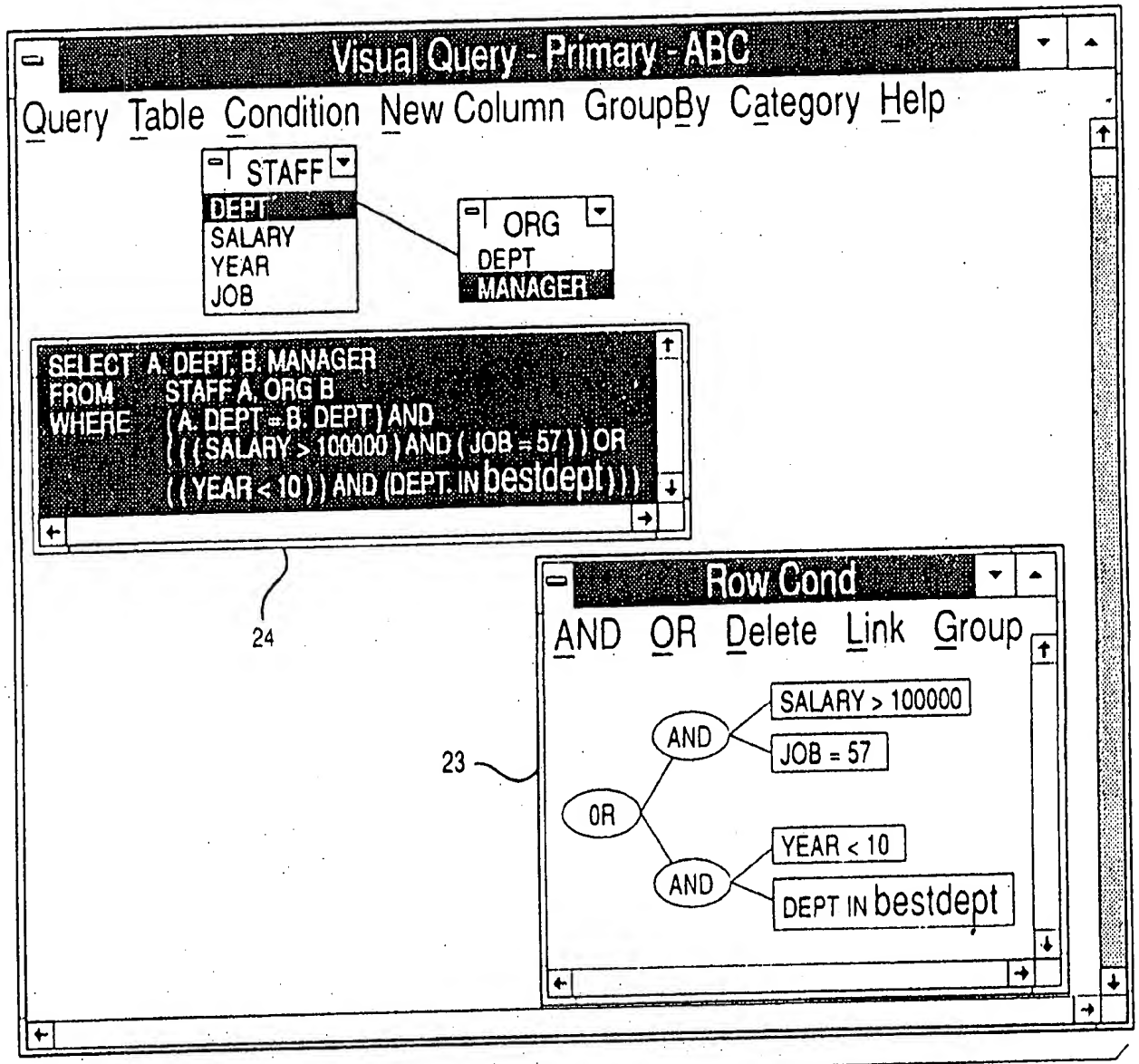


FIG. 5E

EP 0 491 517 A2

(1) The "AND", "OR" nodes can be changed by:

(a) Click on the operator node ("AND", "OR"), the node will be highlighted

(b) Click on AND or OR in the action bar.

—— Notice the 3 operator nodes have all been changed from FIG. 5E, and the corresponding SQL window reflects this change in the SQL select statement.

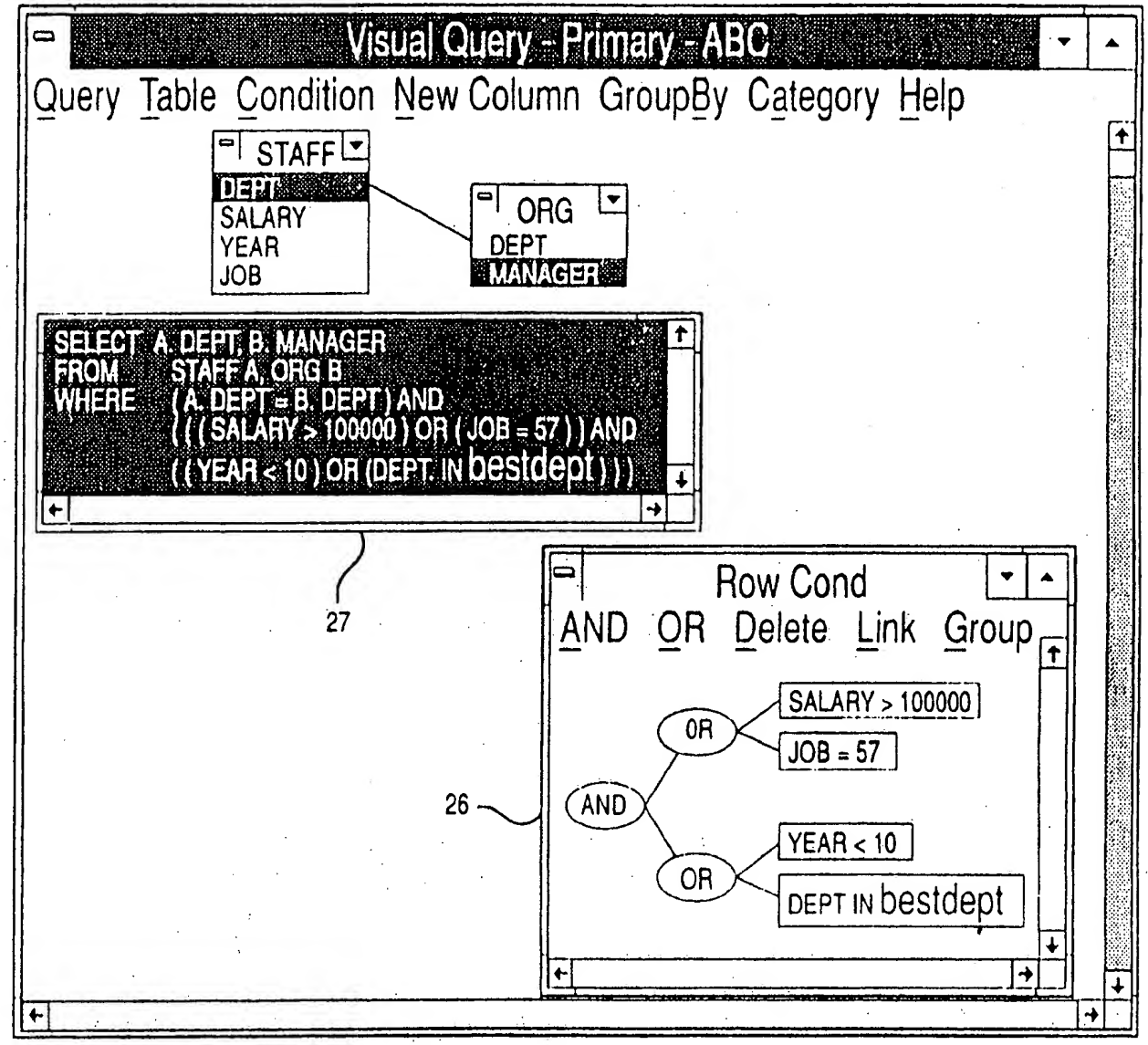


FIG. 5F

(1) A new predicate "MANAGER Not In newmanager" is defined.

(2) It has no logical relationship with any other predicate yet.

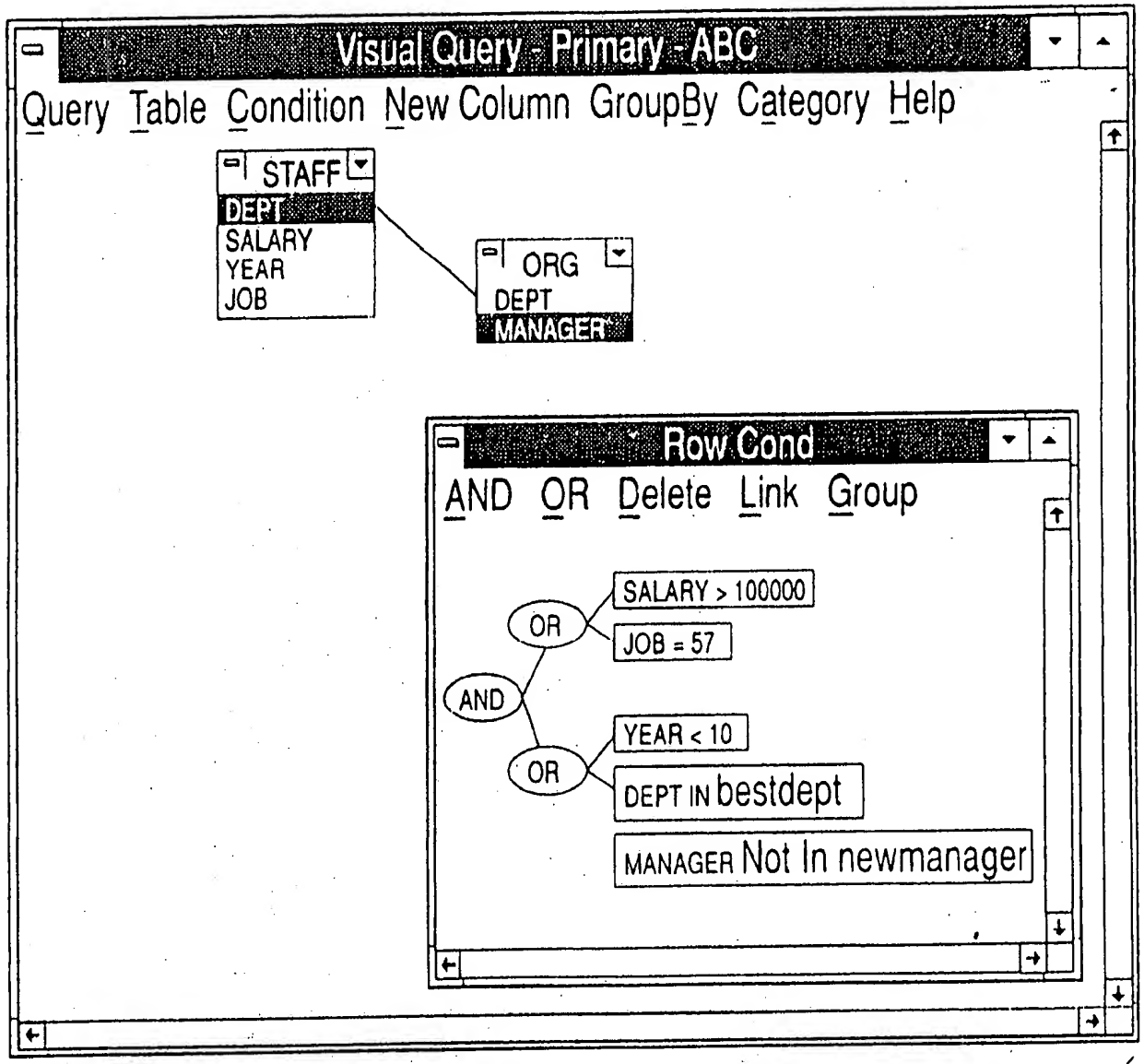


FIG. 5G

EP 0 491 517 A2

(1) This newly defined predicate is "OR" linked with "YEAR < 10" and "DEPT IN bestdept" by:

ⓐ Select "MANAGER Not In newmanager" by clicking on it.

ⓑ Click on Link in the action bar.

ⓒ Click on the target node. (In this example, it is the lower "OR" node.)

—— Alternative for ⓐ + ⓑ is: drag "MANAGER Not In newmanager" and drop on the lower "OR" node

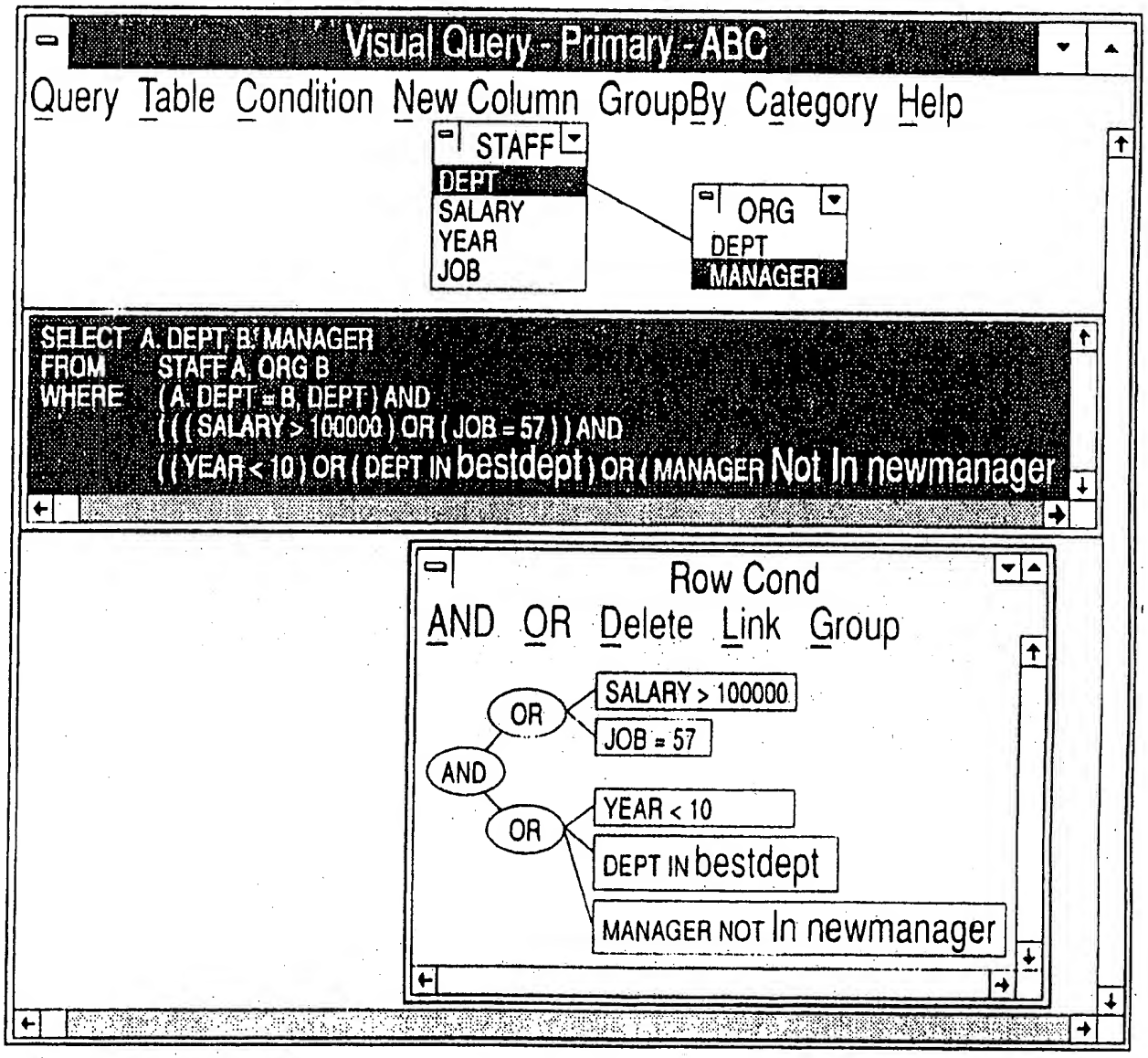


FIG. 5H

EP 0 491 517 A2

(1) Re-grouping: By using the "Link" operation described in FIG 5H, the predicates can be re-grouped. In this example, the "MANAGER Not In newmanager" and "JOB = 57" are switched.

—— Notice the corresponding SQL select statement in the SQL window has been changed accordingly.

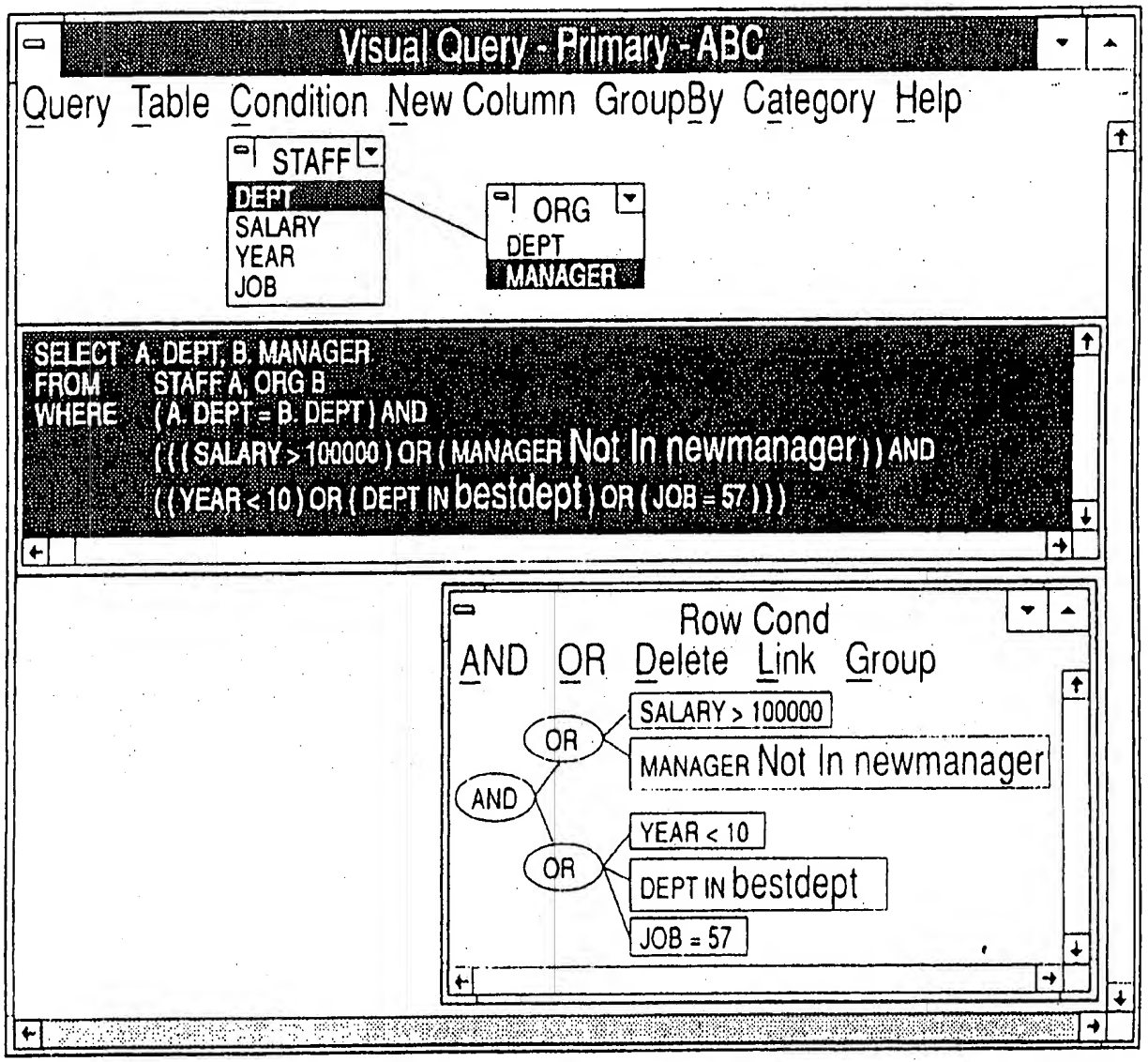


FIG. 5I

EP 0 491 517 A2

FIG. 6A

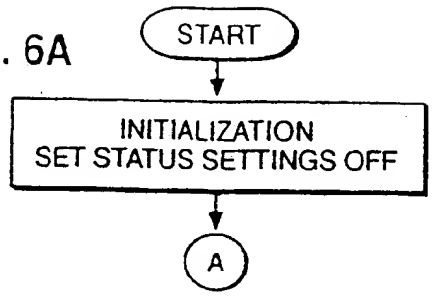
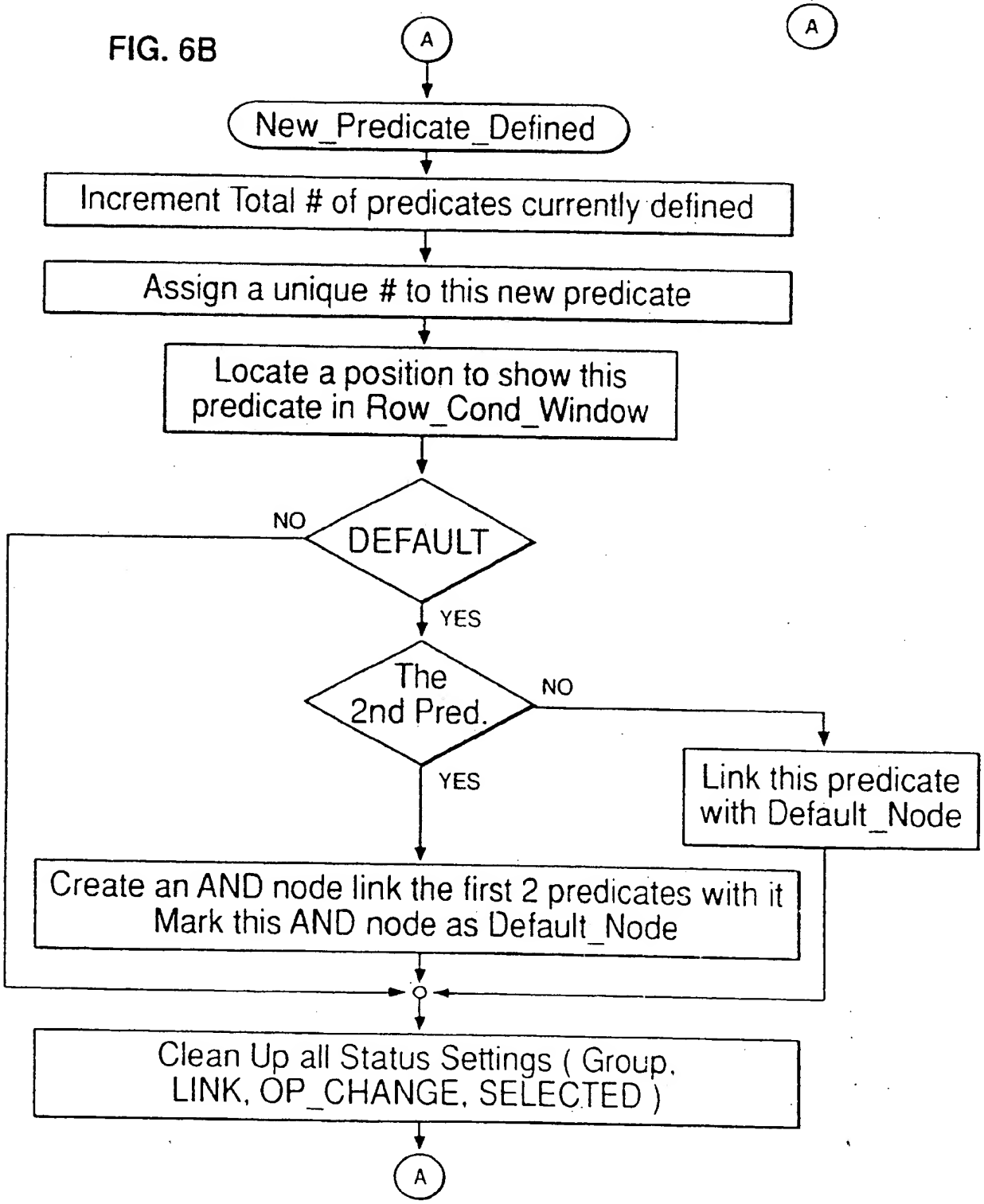

START
INITIALIZATION
SET STATUS SETTINGS OFF
A

FIG. 6B


A
New_Predicate_Defined
Increment Total # of predicates currently defined
Assign a unique # to this new predicate
Locate a position to show this predicate in Row_Cond_Window
NO
DEFAULT
YES
The 2nd Pred.
NO
YES
Link this predicate with Default_Node
Create an AND node link the first 2 predicates with it
Mark this AND node as Default_Node
Clean Up all Status Settings (Group, LINK, OP_CHANGE, SELECTED)
A

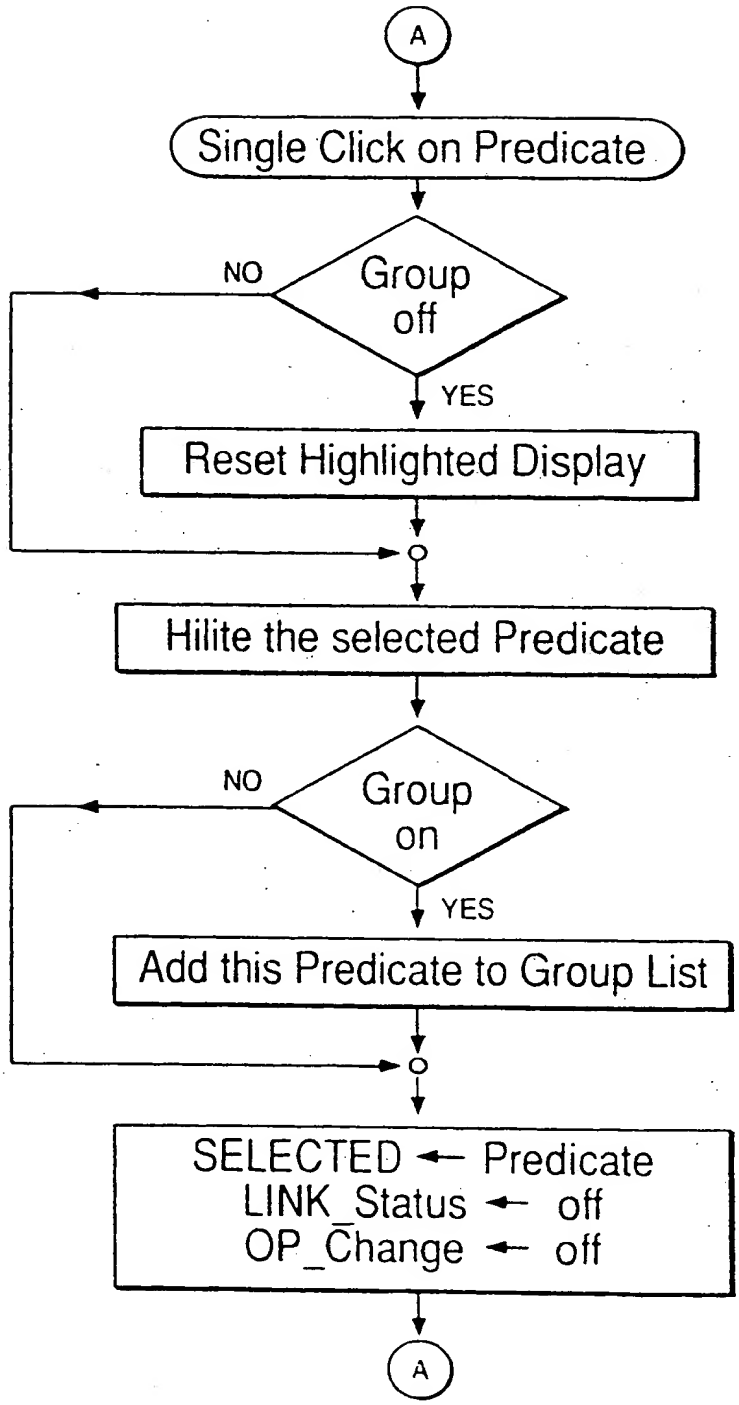

A
Single Click on Predicate
NO
Group off
YES
Reset Highlighted Display
Hilite the selected Predicate
NO
Group on
YES
Add this Predicate to Group List
SELECTED ← Predicate
LINK_Status ← off
OP_Change ← off
A

FIG. 6C

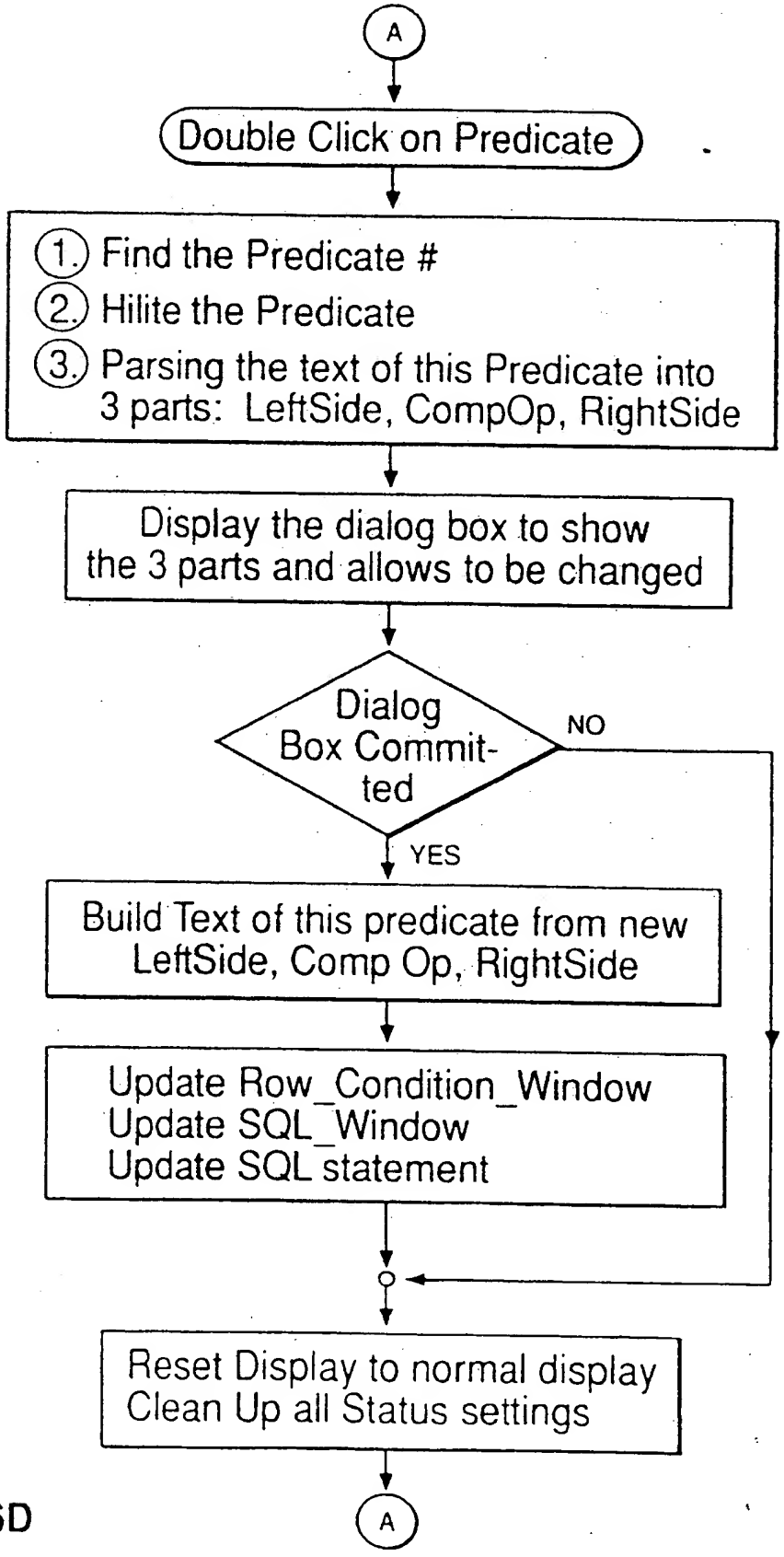

A
Double Click on Predicate
1. Find the Predicate #
2. Hilite the Predicate
3. Parsing the text of this Predicate into 3 parts: LeftSide, CompOp, RightSide
Display the dialog box to show the 3 parts and allows to be changed
Dialog Box Commit-ted
NO
YES
Build Text of this predicate from new LeftSide, Comp Op, RightSide
Update Row_Condition_Window
Update SQL_Window
Update SQL statement
Reset Display to normal display
Clean Up all Status settings
A

FIG. 6D

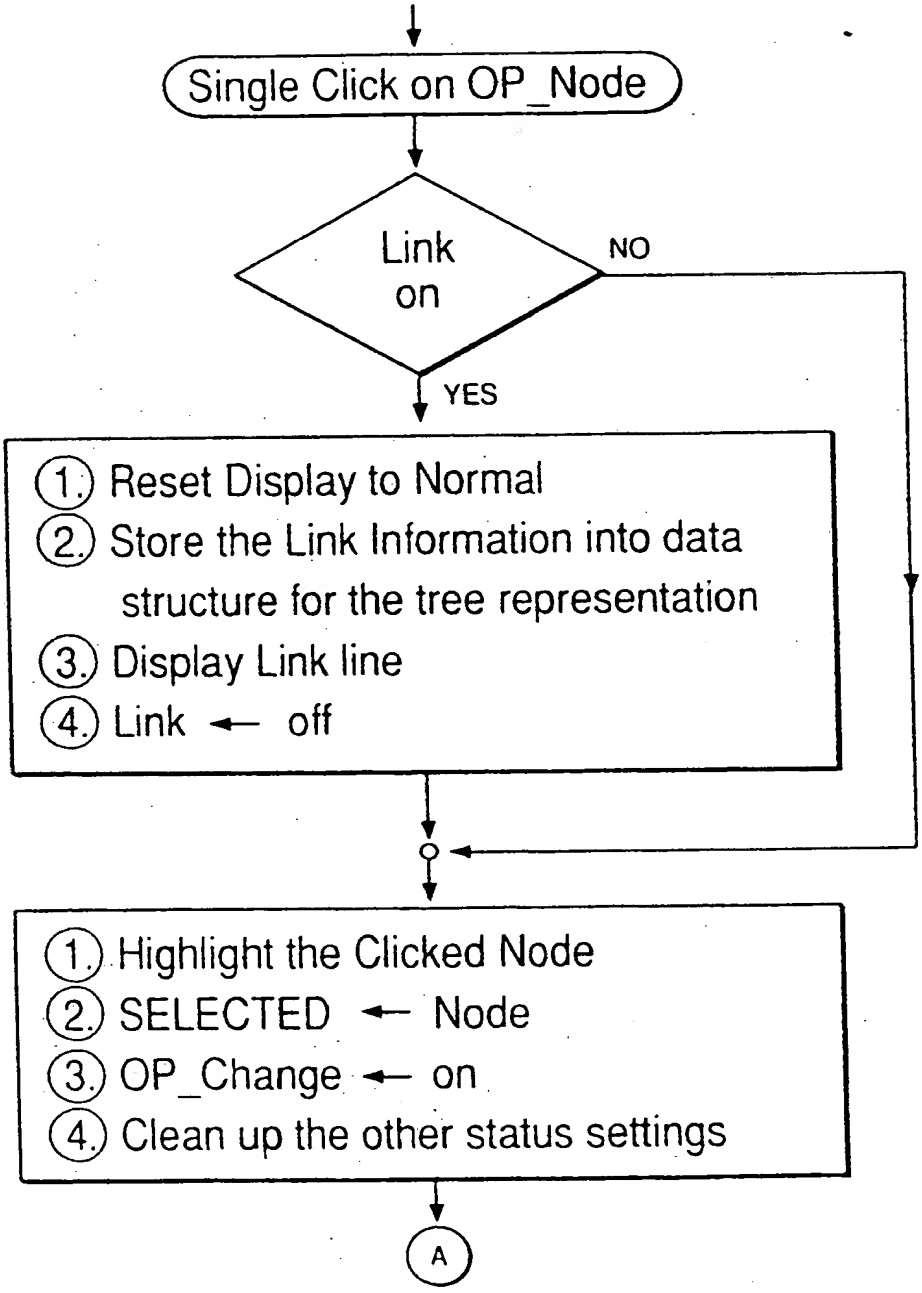
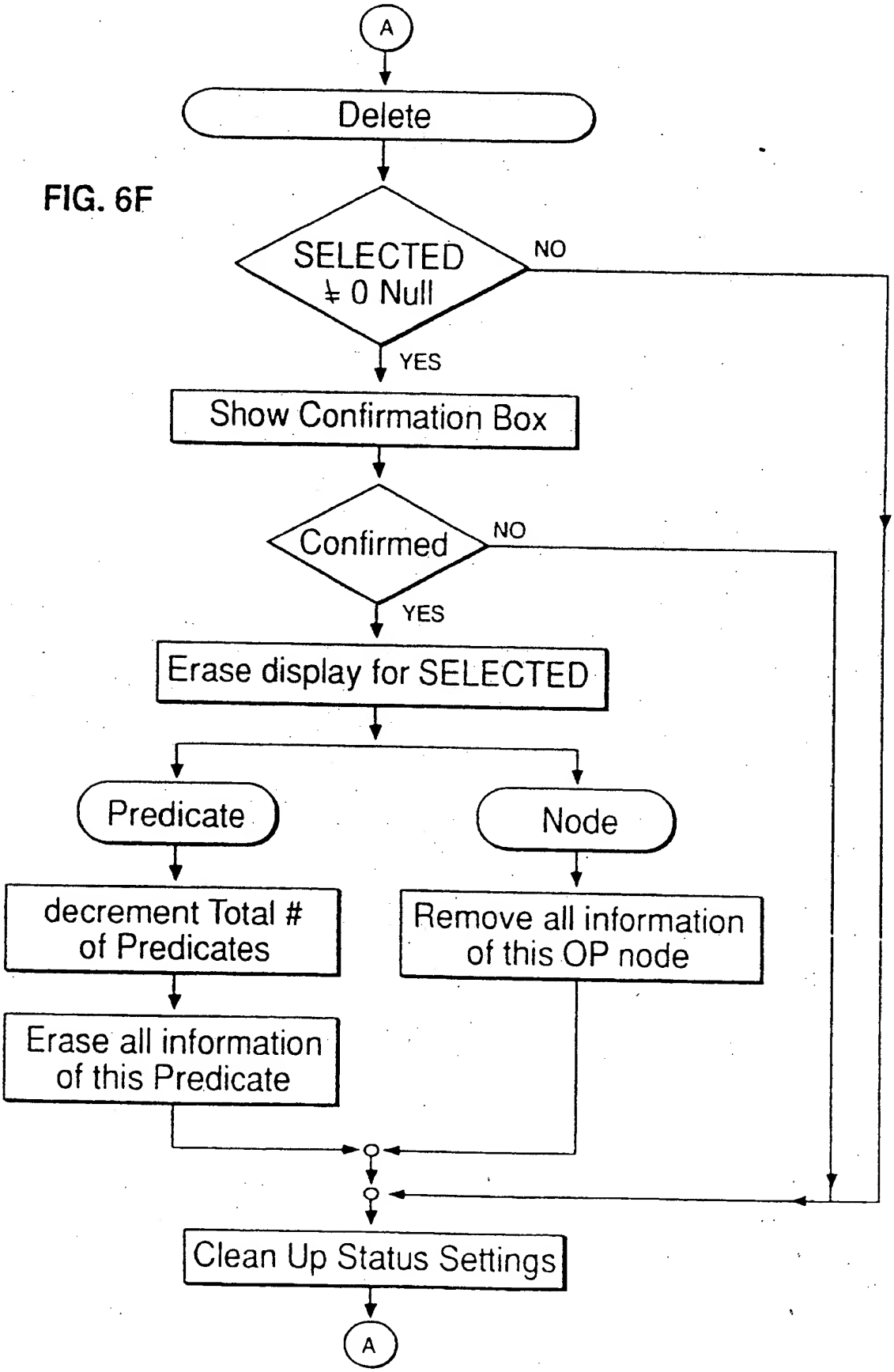

Single Click on OP_Node
Link on
NO
YES
1. Reset Display to Normal
2. Store the Link Information into data structure for the tree representation
3. Display Link line
4. Link ← off
1. Highlight the Clicked Node
2. SELECTED ← Node
3. OP_Change ← on
4. Clean up the other status settings
A

FIG. 6E

FIG. 6F


A
Delete
SELECTED ≠ 0 Null
NO
YES
Show Confirmation Box
Confirmed
NO
YES
Erase display for SELECTED
Predicate
Node
decrement Total # of Predicates
Remove all information of this OP node
Erase all information of this Predicate
Clean Up Status Settings
A

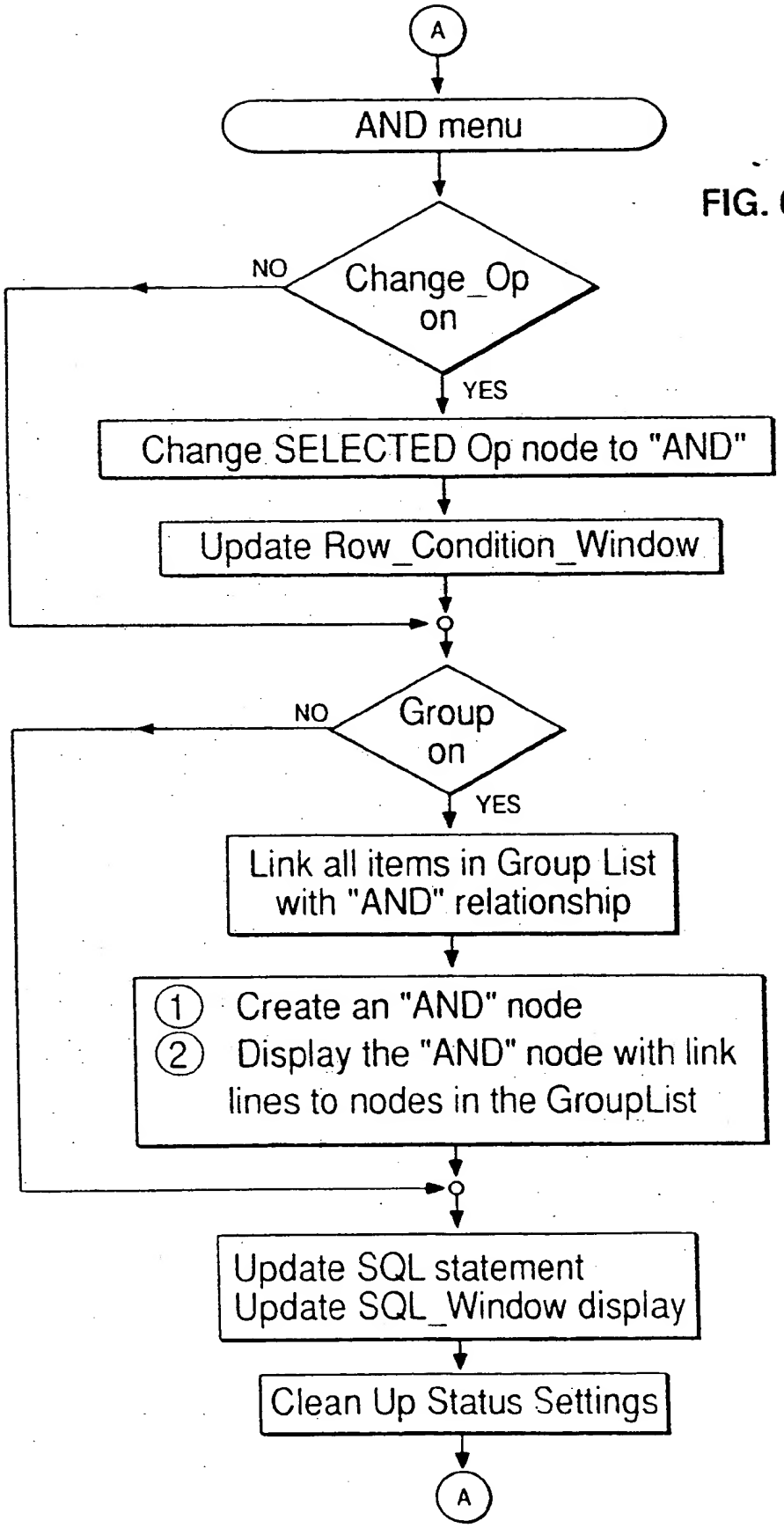

A
AND menu
NO
Change_Op on
YES
Change SELECTED Op node to "AND"
Update Row_Condition_Window
NO
Group on
YES
Link all items in Group List with "AND" relationship
1 Create an "AND" node
2 Display the "AND" node with link lines to nodes in the GroupList
Update SQL statement
Update SQL_Window display
Clean Up Status Settings
A

FIG. 6G

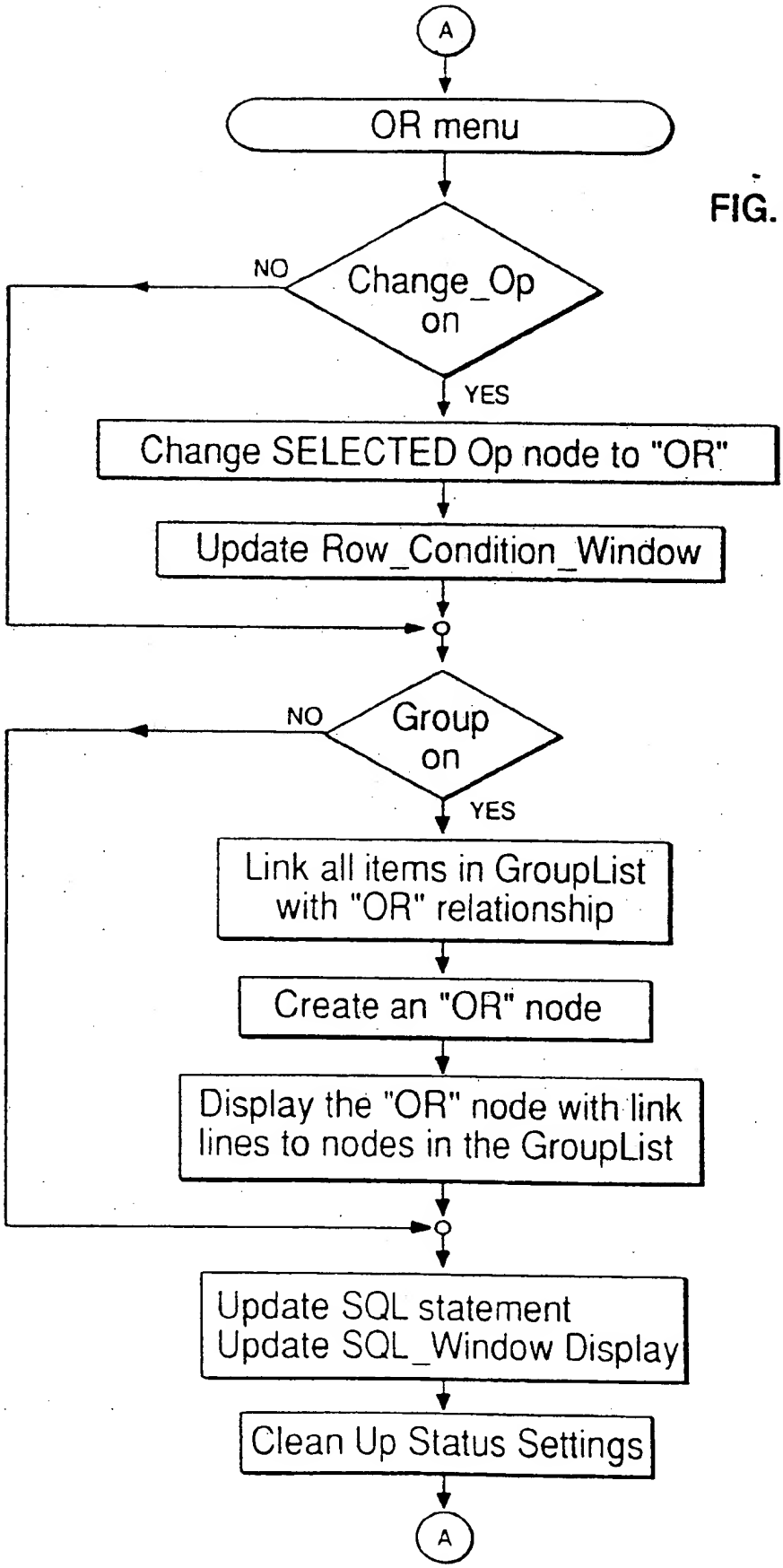

A
OR menu
NO
Change_Op
on
YES
Change SELECTED Op node to "OR"
Update Row_Condition_Window
NO
Group
on
YES
Link all items in GroupList
with "OR" relationship
Create an "OR" node
Display the "OR" node with link
lines to nodes in the GroupList
Update SQL statement
Update SQL_Window Display
Clean Up Status Settings
A

FIG. 6H

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 91 31 1451

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
A	1988 IEEE WORKSHOP ON VISUAL LANGUAGES 10 October 1988, PITTSBURGH, US pages 14 - 20 CZEJDO B. ET AL : 'Design and Implementation of an interactive graphical query interface for a relational database management system' * page 17, column 1, line 1 - page 19, column 2, line 1; figures 1-3 *	1,6,10	G06F15/40 G06F15/403
A	IEEE SOFTWARE vol. 7, no. 6, November 1990, LOS ALAMITOS US pages 63 - 68 T. ICHIKAWA ET AL : 'Iconic Programming: Where to Go ?' * page 64, column 3, line 10 - line 43; figure 2 *	1,6,10	
D,A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 32, no. 5B, October 1989, NEW YORK US pages 368 - 375 'Method of detecting atomic boolean factors'	1,6,10	TECHNICAL FIELDS SEARCHED (Int. Cl.5) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	21 APRIL 1993	FOURNIER C.D.J.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P0401)